User's Manual

TimeStorm®

Version 4.4.0

Embedded Linux from a Trusted Source

timesys®

Contents

TimeStorm Overview

TimeStorm is an Eclipse-based integrated development environment (IDE) for embedded application development. TimeStorm 4.4.0 is based on Eclipse Kepler (Eclipse 4.3.0 and C/C++ Development Tools 8.2.0). TimeStorm supports:

- C/C++ application development
- Qt application development
- Kernel Development
- Kernel Module Development
- Toolchain Management
- Remote Target Management
- Remote Target Console
- Remote Run / Debug of applications
- QEMU
- Remote System Explorer
- Gprof
- Gcov
- LTTng
- OProfile
- Valgrind

Installing TimeStorm

Host System Requirements

Requirement	Linux
Operating System Version	• Fedora 18 • Ubuntu 12.04 • Other recent 32-bit or 64-bit Linux distributions
Memory	1 GB
Disk Space	250 MB

What TimeStorm Installs

During the initial installation, the TimeStorm IDE installs everything it needs to run. TimeStorm is written in Java™ and native code, so in order to ensure that no incompatibilities arise when using TimeStorm with the default version of Java on the host machine, TimeStorm installs a Java Runtime Environment™ that it alone uses.

In order to compile programs, TimeStorm must have a toolchain. By default, TimeStorm uses the toolchain on the system. LinuxLink Starting Points as well as builds using the LinuxLink Factory build system provide cross-compilation toolchains. The installation process for a LinuxLink Starting Point puts the toolchain and any other supporting files in the correct location, no additional configuration are necessary. Working with Toolchains is described later in this guide.

In order to communicate with the target, TimeStorm must be able to communicate with the board using one of several methods. TimeStorm can use telnet/ftp or ssh/scp to communicate with the board, and it has been tested using busybox telnetd, inetutils telnet/ftp servers and the dropbear ssh/scp servers. As an alternative, if the previous network services are not available, TimeStorm can communicate with the board over a connection to the serial console and copy files into a local NFS share.

In order to use TimeStorm, it must have a valid license. The license management is described in the following chapter.

License Management

After installing TimeStorm software, you must install the software license file that enables the features that you have purchased.

TimeStorm 4.4 uses node-locked licenses which are text files with a .lic file extension. A node-locked license enables TimeStorm features on one machine only and is restricted to the hardware address of the machine. The licenses are keyed to the MAC address of the system. Once a license is installed and

used, it cannot be moved to a machine with a different MAC address. If needed, you can move the license to a different machine by creating a new license and installing it on the new machine.

NOTE: *Since the TimeStorm 4.4.0 license system is based on a different technology than the TimeStorm 3.x series of products, you must obtain a new license for TimeStorm 4.4.0.*

Creating a TimeStorm License

To create a license for TimeStorm tools, you must have a LinuxLink seat assigned to you.

- You can create a new license for TimeStorm three times within the life of the subscription.
- If you need to create additional licenses beyond this limit, contact Timesys. These requests are handled on a case-by-case basis.
- ***Team Manager —*** If you are the Team Manager of your LinuxLink account, you can create a license for all the seats within your account.
- ***Developer*** — If you are a Developer, you can create licenses for the seats assigned to you.

To create a TimeStorm license:

1. Note the MAC address of the machine on which you will be using TimeStorm. You may use any of the network interfaces, however if you are running TimeStorm in a Virtual Box environment, make sure you use the MAC address of the NIC inside the virtual machine.
2. Log into your LinuxLink account (https://linuxlink.timesys.com/) via a web browser, and then:
    - If you are a ***Team Manager***, click on the team name at the top of the page.
    - If you are a ***Developer*** click on the user name at the top of the page.
3. Scroll to the bottom of the page, and click the 'Edit Licenses' button in the Active Licenses area.
4. Next, click the 'Create' button located to the right of the user of the seat for which you want to create the license.
    - Enter the MAC address and a descriptive name for the license. It can be helpful to include the type of operating system and computer that will use the license.
    - The license expiration date and the user's email address are entered automatically.
    - Choose to create a license for TimeStorm 4.x. ***NOTE: Licenses generated for TimeStorm 3.x will not work with TimeStorm 4.4.0,*** *so be sure to select the correct version when generating the license.*
    - Click 'Next' to generate the license. You are given the option of editing the CC field and the text of the message that will be used to send the license file.

- Click 'Send.' An email with the attached license is sent, and the contents of the license file are displayed.
- Save the license on the computer you will be using for TimeStorm. LinuxLink will keep the license on file, so it can be retrieved at any time.

Installing the License

Installing a license involves copying your license file into the appropriate directory on the TimeStorm host. License files are typically delivered as text file attachments to email messages from Timesys. You can rename a license file, but you must retain the .lic file extension. To install the license locally, you can do one of the following:

- From within the 'Create a License' window, click the Download link and save the license file on the TimeStorm host at a location described below.

  OR

- Save the emailed license attachment on the TimeStorm host to one of the locations described below.

At startup time, TimeStorm will automatically detect license files if they are stored locally on the TimeStorm host in one of three locations/directories. The order and locations in which TimeStorm checks for license files are outlined, below:

1. **<user home directory>/.timesys/timestorm/licenses**
   The TimeStorm user's home directory is typically something like `/home/<username>` on Linux systems.
   - Licenses copied to this location will work for this user for all version-compatible TimeStorm installations on this computer.
2. **<TimeStorm installation directory>/licenses**
   TimeStorm checks for a directory named 'licenses' that is located within the TimeStorm installation directory.
   - This licenses directory is created automatically when TimeStorm is installed. Licenses installed in this directory will work for this installation of TimeStorm only.

Installing TimeStorm

TimeStorm is available in 32 bit and 64 bit versions. Depending on whether you are running a 32 bit or a 64 bit OS, select and download the 32 bit or 64 bit version of TimeStorm.

To install TimeStorm:

1. Open a terminal and change directory to the location where you want to install TimeStorm.

2. Use the following command to uncompress the archive on your host, and extract its contents (the build number <build_num>, will vary):

   ```
   # tar -zxf timestorm-full-install-4.4.0.<build_num>.tgz
   ```

3. TimeStorm is installed within the directory "timestorm-4.4.0". You can start TimeStorm with the following command:

   ```
   # ./timestorm-4.4.0/timestorm
   ```

TimeStorm and Eclipse

TimeStorm is based on the Eclipse IDE, first published by IBM and now maintained by the Eclipse Foundation. TimeStorm is designed to conform to the Eclipse standards as closely as possible, using as much of the existing user interface language as possible.

This section defines the basics of the Eclipse environment and explains how Timesys has extended Eclipse by adding features that make TimeStorm unique to embedded developers.

Eclipse Terminology

Workspace

The workspace is the top-level container for all of the information kept by TimeStorm. When starting TimeStorm, the user selects a workspace and uses only that workspace. Users may have more than one workspace, but TimeStorm uses only one at a time. When using TimeStorm, users can switch workspaces.

Projects

Projects in TimeStorm are different than workspaces in that the workspace contains projects, and projects are the entity responsible for creating binaries. A workspace frequently contains several projects: one for building the application binaries, another for creating a library, etc. Using the workspace, these projects can be coordinated to produce the software binaries for the target board.

Wizards

Every project in TimeStorm is created with a wizard. Wizards in TimeStorm ask the user for basic project information and will create a simple, working project that is used as the base for additional work.

Workbench

The TimeStorm IDE graphical user interface is called the Workbench. Workbench features are part of the standard Eclipse development environment, which serves as the basis for TimeStorm IDE. The Workbench is composed of editors, views and perspectives.

The Eclipse Workbench is presented into one or more windows. These windows could be either views or editors, as shown in *Figure 1*. A perspective defines the visual arrangement of the Workbench windows.

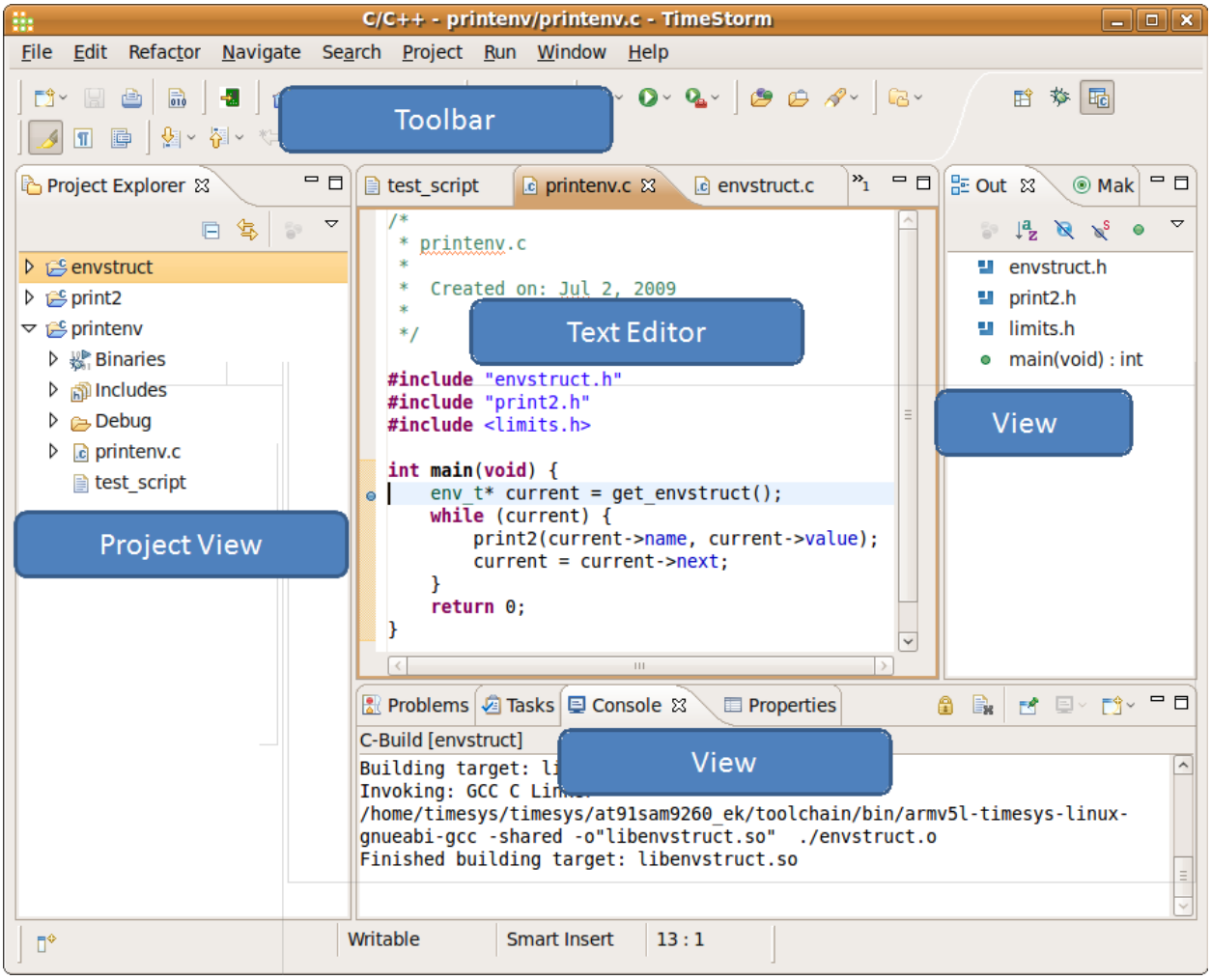


Figure 1: TimeStorm Workbench

The Workbench is documented more completely in the online Workbench User Guide, which is available from the TimeStorm Help menu.

Views and Editors

Views support editors and provide alternative presentations as well as ways to navigate information in your workspace. For example, the Project Explorer view *(shown in Figure 1, above)* displays projects in the current workspace. Views have their own menus, and some views have their own toolbars. The actions represented by buttons on view toolbars affect only the items within that view.

Editors are stacked in the center. You can associate different editors with different types of files. Double-clicking a file to open it will open the associated editor in the workbench *(shown in Figure 1, above)*.

Perspectives

You can rearrange the views and editors in your workbench as you prefer. The saved arrangements of views and editors are called Perspectives. There are no constraints as to what views appear in a perspective; however, the views are typically related in some way. TimeStorm ships with several perspectives, for example:

- **C/C++**: Views and menu items that are customized for applications, libraries and driver development
- **Debug**: Shows debugging controls (step in, step over, step return, terminate, etc.) and information (variables, breakpoints, threads, etc.)
- **Qt C++:** Includes views for Qt Widget box, Action Editor, Signal Slot Editor, etc.
- **Git Repository Exploring:** Includes explorer view for adding, browsing and managing Git repository

When TimeStorm initially opens, it loads the C/C++ perspective by default. If you create a new project of a different type, TimeStorm switches to the appropriate perspective automatically. By default, TimeStorm saves your current perspective settings when you exit TimeStorm and reestablishes it when you start again. If you want the workbench to revert to the standard layout for the current perspective, use the *Window > Reset Perspective* menu.

Based on working preferences, perspectives can be changed and/or additional perspectives can be created by the TimeStorm user.

Source Code Control Systems

TimeStorm relies on the Eclipse framework to provide integration with source code control (SCC) systems. The open nature of Eclipse has resulted in integration of many source code controls systems. TimeStorm bundles Egit that work with git. TimeStorm users are free to select a SCC that best matches their needs.

Using Your Own Source Code Control System

TimeStorm can be used with any source code control systems. If the source code control system in use at your company isn't directly supported by TimeStorm, you can still use it to manage your development tasks. There are two strategies, using a vendor supplied plug-in or doing source code control from the command-line.

Using Source Code Control Plug-in

Many vendors have plug-ins that can be added to an existing TimeStorm installation. Since TimeStorm is built on Eclipse, a plug-in that is compatible with the Eclipse used to make TimeStorm can be used without modification. Follow the directions supplied by the vendor to add the plug-in to TimeStorm.

Using Source Code Control Command Line

Some source code control systems do not have Eclipse-based plug-ins. You can still use these, but you'll need to do so from the command-line. When using a SCC command-line tool, follow these guidelines

- **DO NOT version control the .metadata directory**
  TimeStorm maintains a *.metadata* directory under the workspace. This directory contains temporary information related to the workspace and should not be checked into version control.

- **Do not make any files starting with "." Read-only**
  TimeStorm creates files in project directories starting with . to store any project related information. These files can be checked into version control, but must read/write on the local file system. Some source code control managers make all files read-only unless otherwise instructed.

- **When initially checking out a project, put in the directory of an existing project.**
  To add the project to the workspace, create a project with the wizard (you can delete all of the files) and use the SCC to do the initial check out of the project files into that directory. After the check out, return to TimeStorm, and refresh the project by pressing F5 which will enable TimeStorm to recognize the newly checked-out files.

Embedded Development with TimeStorm

Embedded development is different from traditional development because:

- The target machine is usually different hardware architecture than the host machine, and
- The target machine lacks the resources necessary to be used as a development machine.

Development Host and Target Board

The development host is the machine where TimeStorm runs and source code is compiled for the target board. The target board is the location where the code that is compiled on the development host will eventually be debugged and deployed. Because, target boards often do not have the resources (like RAM or fixed storage), processing power or peripherals (such as a monitor, keyboard or network connection) to support a development environment, engineers typically don't do their development directly on the target.

TimeStorm bridges the gap between the more powerful development host and less powerful target board through the following features:

- **Toolchain Management** – TimeStorm keeps track of any toolchains that are installed on the development host as part of a Timesys Starting Point or are created with the Factory build system.

  > ***NOTE:*** *A '**starting point**' is Timesys' terminology for a pre-built BSP/SDK. A starting point is a platform and toolchain that has been specified by Timesys and is pre-built and available for download from the Timesys LinuxLink Web site.*

- **Build Configurations** – Build Configurations connect a toolchain with a project. Each project can have many build configurations and each build configuration can have its own settings to control how the software is built. By default, TimeStorm creates three build configurations for new projects: 1.) Release, 2.) Debug and 3.)Profile.
- **Target Management –** TimeStorm includes a tool for communicating with a target, enabling target information to be stored in one location and shared by those features needing to reach a remote machine.
- **Remote Run/Debug Configurations** – In order to run and debug the code created for the target, TimeStorm includes Run and Debug configurations that use the targets defined by the user to download code and properly configure the run-time environment of the target.

Working with Toolchains

Toolchains include the tools required to compile your applications. Toolchains that you use with TimeStorm for remote development allow you to compile on one platform your applications for use on another platform. They include appropriate cross-compilers, debugging software, and other utilities that

work with your target processor. Build configurations for TimeStorm projects specify which toolchain to use when building your project. You can set different toolchains in different build configurations.

Timesys Toolchains

TimeStorm is designed to automatically detect Linux toolchains from Timesys Starting Points that have been installed using the install shell script as the "root" user.

To display the list of toolchains recognized by TimeStorm:

1. From the main menu, select *Window > Preferences*.
2. From within the Preferences panel, select *TimeStorm > Toolchains* as shown in *Figure 2*.
3. To View the toolchain properties, select the toolchain and click the 'View' button.
   NOTE: *Timesys toolchains that are automatically detected cannot be edited. If you want to modify tools or properties for a Timesys toolchain, you must add it as a user-defined toolchain.*

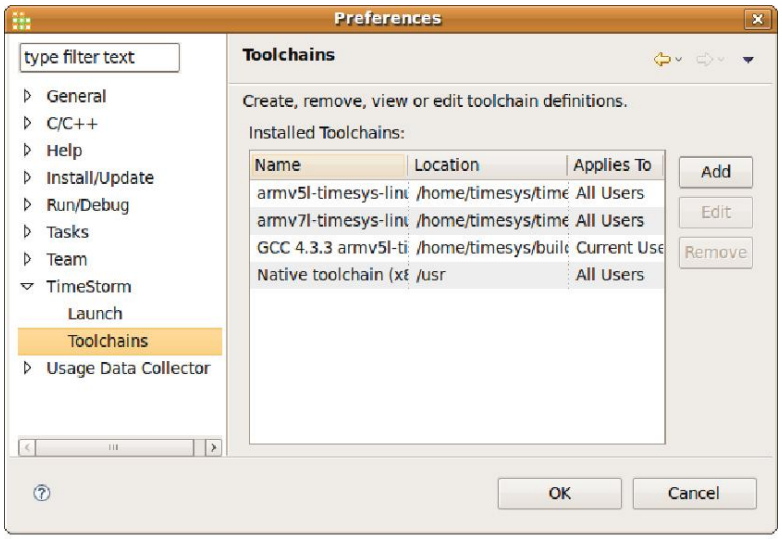


Figure 2: Toolchain Management

NOTE: *Both automatically detected and any added user-defined toolchains will appear in this list.*

User-Supplied Toolchains

If your toolchain is not from a Timesys Starting Point, you must import your toolchain to make it available to TimeStorm. The next section, titled 'Adding Your Own Toolchain,' gives detailed instructions for adding a user-defined toolchain to your TimeStorm Workspace so that it will be available for use in TimeStorm projects.

When using a toolchain that was not provided by Timesys, you must ensure the toolchain's compatibility with TimeStorm and with the other elements of your development environment. The toolchain must be GNU-based, which ensures that cross-compilers and other required utilities are available on the

appropriate path and have the correct permissions for access by the TimeStorm user. In order to perform remote debugging in TimeStorm, the toolchain must include GDB version 5.2.1 or later.

Adding Your Own Toolchain

To add a new user-defined toolchain:

1. Click the 'Add' button (*shown in Figure 2*) to invoke the 'Add Toolchain' wizard.
2. Within the Add Toolchain wizard *(Figure 3*), specify where the toolchain is installed. Use the 'Browse' button to specify the directory that contains the toolchain's binary files.

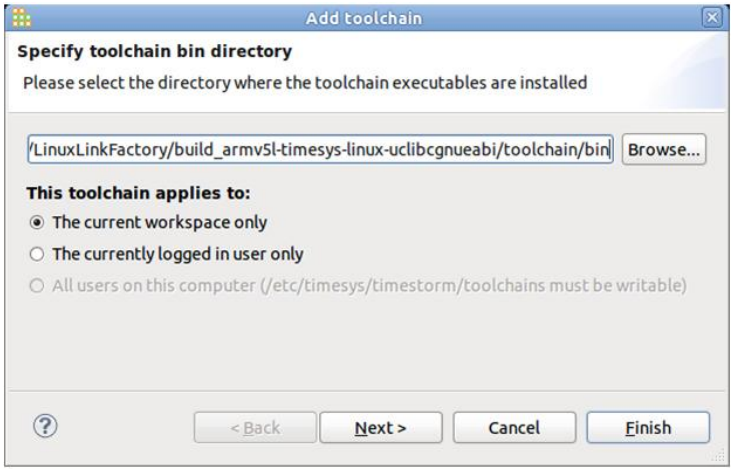


Figure 3: Toolchain Management, Toolchain Directory

3. Click the 'Next' button.

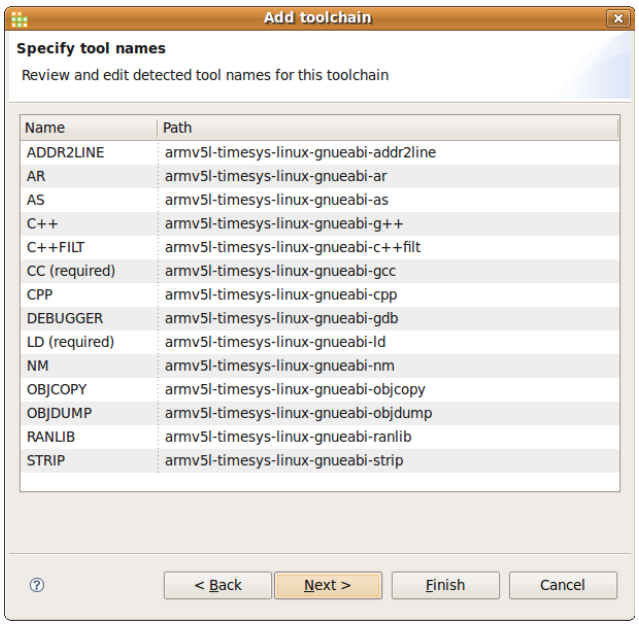


Figure 4: Toolchain Management, Detected Tools

Figure 4 on the previous page shows the list of tools that have been detected in the specified directory. Toolchains using gcc with a cross-compiler prefix will be automatically detected.

4. Modify /Edit the toolchain's name and description (optional)

   Each tool chain has a Name, Description, ID and a location. TimeStorm uses the toolchain identifier (ID) as an internal reference; the name and description are used to identify the toolchain to users. You can customize the name, description, and identifier for this toolchain. Identifiers and names must be unique; that is, two different toolchains on one system cannot have the same identifier or the exact same name.

   The next panel *(shown in Figure 5)*, allows you to modify the toolchain's name and description. If your toolchain has acceptable default values, modifying the name and description information is optional.

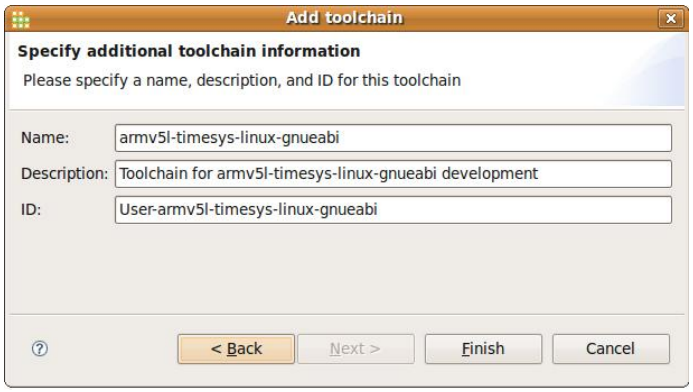


Figure 5: Toolchain Management, Name and ID

Managing Build Configurations and Managed Makefiles

TimeStorm uses build configurations to control how TimeStorm C and C++ projects are compiled and built. A build configuration is a collection of build settings associated with a project that contains the following information.

- **Compiler** – What compiler to use to build the software. TimeStorm projects have the added benefit of enabling the user to create Build Configurations that use a cross-compiler.
- **Compiler Settings** – Settings such as the optimization level, ANSI compliance and warnings.
- **Linker Settings** – Settings that specify what libraries to use and where they can be found.

By switching from one build configuration to another, you can quickly and easily change the build settings for your project.

Types of makefile Management

TimeStorm has two different ways of managing the make files in a project, as outlined below.

1. **C/C++ Executable, Shared Library, and Static Library Projects** — The make file for these projects are generated using the settings from the project build configurations. These projects may use a TimeStorm Cross-Compile Toolchain.

2. **Makefile projects** — In these projects, the make file is under complete control of the user. TimeStorm does not attempt to examine the project and generate a make file before the build. Use this option when working with projects that already have an existing make file, or when complete control of the make file is necessary.

Build configurations are part of the project properties. To see the build configurations for a project, right-click on a project, select 'Propertie,' and then select *C/C++ Build > Settings*. The current build configuration will appear on the right side of the dialog. Use the 'Manage Configurations' button to add, rename or delete build configurations.

Building with Host Compiler

Since TimeStorm allows the user to create multiple build configurations for a project, users will frequently create a build configuration that executes the host compiler (the compiler that produces code to be executed on the host machine) so that the program can be tested without the additional overhead involved with remote debugging. Since most problems are algorithmic in nature, this strategy reduces the amount of development time and is an oft-used strategy used by embedded developers.

To access the build configuration panel:

- Right-click on the project, and select 'Properties.'
- In the properties panel, select *C/C++ Build > Settings*.

To use the toolchain on the host machine:

- Select the 'Cross Toolchain' tab in the build configuration panel then select the 'Native Toolchain' option.
- The 'Native Toolchain' will be appended with text describing the type of platform (for example, '(i686-linux)').
- When using the host compiler, ensure that all of the libraries, both inside of the workspace and toolchain have been compiled for the host platform.

Working with Hardware Targets

Before you can run and or debug on a remote target, TimeStorm will require information about the target(s). TimeStorm uses this information to determine how and where data should be copied to the target, and the method of communication that TimeStorm should use between the host and the target. This is done through the Hardware Targets window.

Target RFS

To download files to the target, the target should be booted with an RFS that includes an FTP server or an SSH server that supports scp, or booted using the NFS share on the host. To communicate with the target, the target should be booted with an RFS that includes a telnet server, or SSH server, or connected to the hosted via the serial port. The pre-built SDK from Timesys includes dropbear that supports both scp and ssh. If you are creating your own SDK using the Timesys FREE Edition (Web Factory), there are many packages from which you can choose for ftp, telnet and ssh. The available packages are listed below.

- **ftp:** proftpd, pure-ftpd, vsftpd, ncftp
- **scp:** dropbear, openssh
- **telnet:** netkit-telnet
- **ssh:** dropbear, openssh

** For tools that include uploading files from the remote target to host (OProfile, LTTng, Valgrind) you should have openssh server running on the target. Dropbear does not include sftp and does not work for uploading files.

Hardware Targets Window

The Hardware Targets window enables you to:

- Add a target to the list of registered targets.
- Edit the information for a registered target.
- Delete a target from the list of registered targets.
- Check connectivity from the host to a target.

You can access the Hardware Targets window either by clicking the Hardware Targets toolbar button 

or by selecting *Run > Hardware Targets* from the main menu, as shown in *Figure 6*, below.

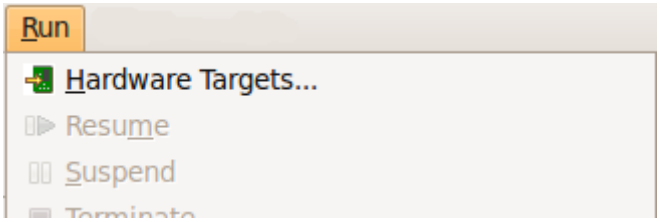


Figure 6: Hardware Targets Menu

Setting up a target requires information for the following three tabs:

TAB 1: Download — How to download files to the target *(shown in Figure 7)*,

TAB 2: Execute — How to communicate to execute on the target *(shown in Figure 8)* and

TAB 3: Raw Log — How to test the connectivity between the host and the target *(shown in Figure 9)*.

TAB 1: Download

TimeStorm allows files to be downloaded to the target in a few different methods. You will be prompted to select one of the options below.

FTP – Files will be downloaded from the host to the target using FTP.

SCP – Files will be downloaded from the host to the target using SCP.

NFS – TimeStorm will copy the files locally in the target Root File System (RFS) mounted over NFS.

None – The host will be connected to the target, but no data will be downloaded. You can then do any needed downloading using the console.

FTP/SCP Settings

The corresponding fields for both the FTP and SCP options *(shown in Figure 7)* are as follows:

***IP Address* —** Enter either the IP address or the hostname of the target. If you use a hostname, your network must include access to a Domain Name Server (DNS) lookup facility.

***User Name* —** Enter the user name for FTP or SCP login on the target, as appropriate.

***Password* —** Enter a non-blank password for FTP or SCP login, as appropriate.

***Destination Directory* —** Enter the path to which the files will be transferred on the target. FTP settings sometimes restrict the location of files copied to a directory under the home directory of the user. Therefore, the actual destination directory depends on the

configuration of your FTP software. In some cases, directories specified in this panel are appended to the user's home directory on the target. Ensure the user name that you specified has read/write permissions to the destination directory.

***Link to Execution* —** When this checkbox is selected, all of the information listed in the fields corresponding to the FTP or SCP option is automatically entered into the fields in the Execute tab. Those fields are not editable in the 'Execute' tab when this checkbox is selected. This checkbox is selected by default. When this checkbox is deselected, you must manually enter the corresponding information in the fields in the 'Execute' tab.

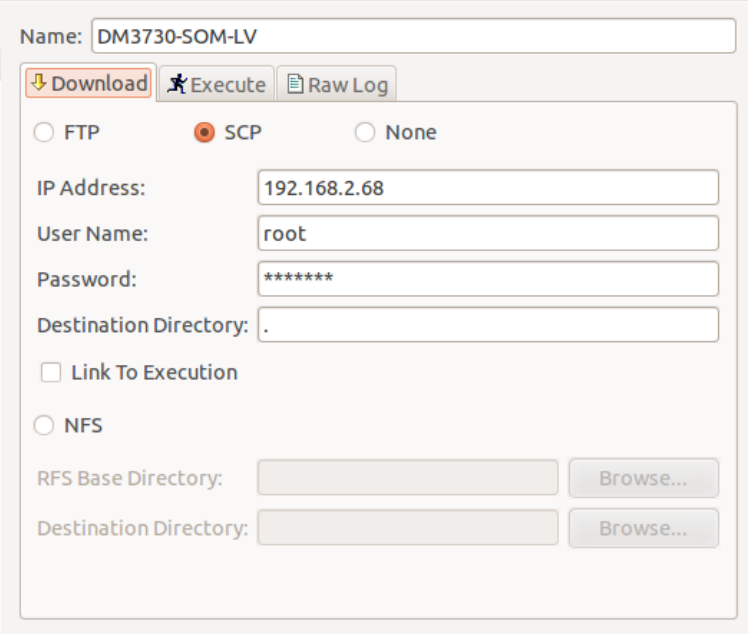


Figure 7: Download Tab

NFS Settings

The corresponding fields for the NFS option *(shown in Figure 7)* are as follows:

***RFS Base Directory*—** Use the Browse button to select the location of the RFS on the host. This location is where the target's root filesystem is mounted over NFS; for example, `/home/user/timesys/boardname/rfs`, where *user* is the current username and *boardname* is the name of the target board.

***Destination Directory* —** Use the Browse button to select the location in the RFS to where the files will be downloaded. The directory must be within the RFS Base Directory location that was selected. Ensure that you have read/write permissions to the destination directory.

After you have entered the required information in this tab, select the 'Execute' tab, and enter the appropriate information.

TAB 2: Execute

Use the Execute tab *(shown in Figure 8)* to specify the communication method that TimeStorm uses between the host and the target.

Within this tab, you can select from the following communication methods between the host and the target:

Telnet – Communication between the host and the target will occur when using Telnet.

SSH – Communication between the host and the target will occur when using secure shell (SSH).

Serial – Communication between the host and the target will occur using the serial connection.

You must fill in all of the corresponding fields for the method that you choose. TimeStorm answers the login and password prompts presented by the Telnet or SSH server on the target, based on the data that you enter in this tab.

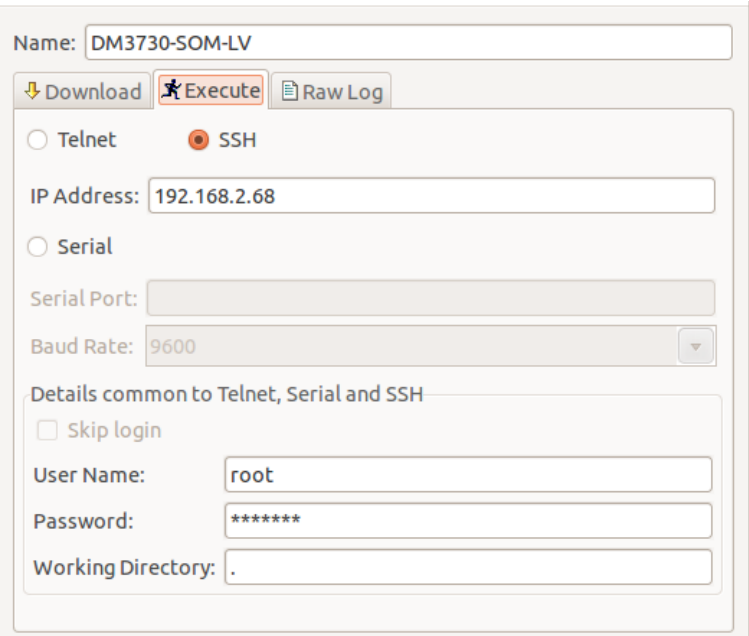


Figure 8: Execute Tab

Telnet/SSH Settings

The corresponding field for both the Telnet and SSH options are as follows:

***IP Address* —** Enter either the IP address or the hostname of the target. If you use a hostname, your network must include access to a DNS lookup facility.

The execution IP Address field is also used when connecting to gdbserver for TCP-based remote debugging. This field must be provided regardless of connection method to use TCP-based debugging.

Serial Settings

The corresponding fields for the Serial option are as follows:

***Skip login* —** If you are connecting directly to the target's bootloader, and thus will not get a login prompt, select this option. The rest of the common details will be grayed out.

***Serial Port* —** Enter the host's serial port; for example, /dev/ttyS0. If you use this option, be sure that the user that you specified has the proper permissions to access the serial port.

***Baud Rate*—** Use the drop-down list to select the appropriate baud rate for serial communication. The default rate is 9600 bps.

Common Settings for All Options

The common fields for all three options in the Execute tab are as follows:

***User Name* —** Enter your user name for a Telnet, SSH, or serial login, as appropriate, on the target.

***Password* —** Enter a non-blank password for the Telnet, SSH, or serial login, as appropriate, on the target. Note that the Timesys pre-built SDK does not set the root password in the RFS. You have to set the password for the root user after the target boots, and enter the password in this text field.

***Working Directory* —** Enter the directory to which control will be transferred after TimeStorm logs in to the target. By default, this is the same path to which the data will be transferred on the target. This directory must already exist.

NOTE: *If the 'Link to Execution' checkbox is selected in the Download tab, the information in the fields corresponding to the FTP or SCP option in the Download tab is automatically entered into the common fields and the IP Address field in the Execute tab. In this case, the fields are not editable as long as the 'Link to Execution' checkbox is selected in the Download tab.*

TAB 3: Raw Log

After you register a target, you can check the connectivity to that target at any time. Each time you add a target or edit information for a target, Timesys recommends that you perform a connectivity check to verify that the host can communicate with the target. The target connectivity check does not occur automatically upon registering or applying changes to a target.

To verify connectivity to a target, select the target in the left panel of the 'Targets' window and click the 'Check Link' button. The Raw Log tab *(shown in Figure 9)* becomes active when the check starts.

If the host communicates successfully with the target, a '`Target Check - Passed`' message *(shown in figure 9)* is displayed in the 'Raw Log' tab of the' Targets' window.

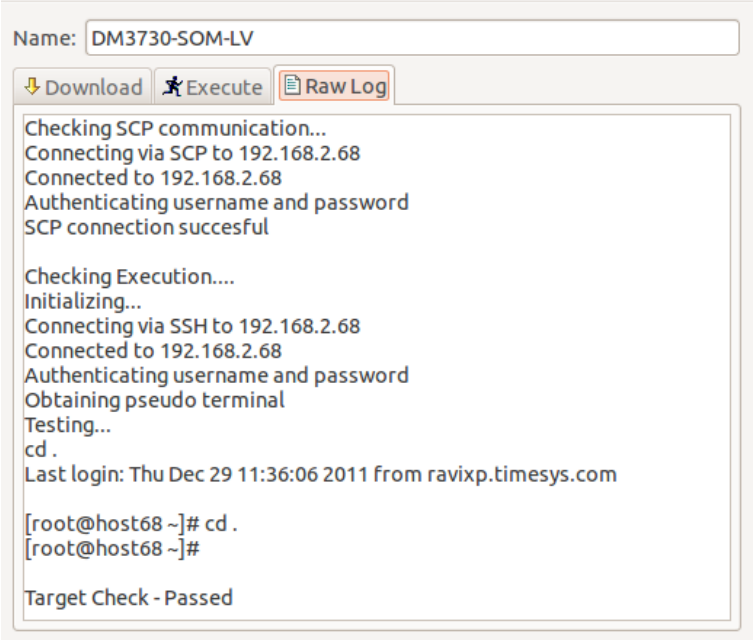


Figure 9: Output from Check Link

Hardware Target Console

When you remotely run or debug an application on your hardware target, a console view is launched and the commands being run on the target and the target's console output are displayed in the view. If you directly want to interact with the target, like run some commands on the target, copy files to/from the target, etc., you can launch a target console by clicking the icon in the toolbar or by clikcing *Run > Hardware Target Console*. This will open the 'Choose a Target' dialog, as shown in *Figure 10*. Select the target you want to open the console, and click 'OK.'

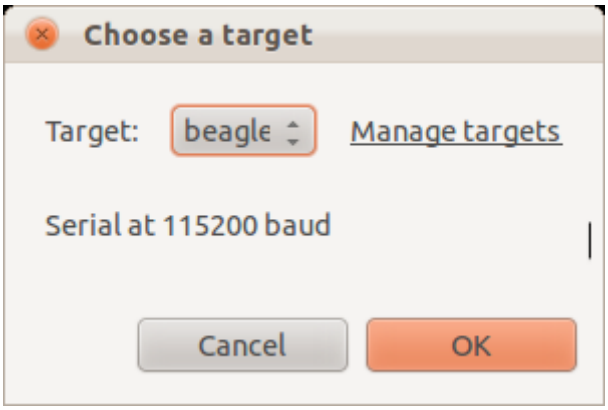


Figure 10: Choose target for target console

TimeStorm connects to the target using the Execution Settings, Telnet, ssh or serial, configured for the target and opens a console view, as shown in *Figure 11*.

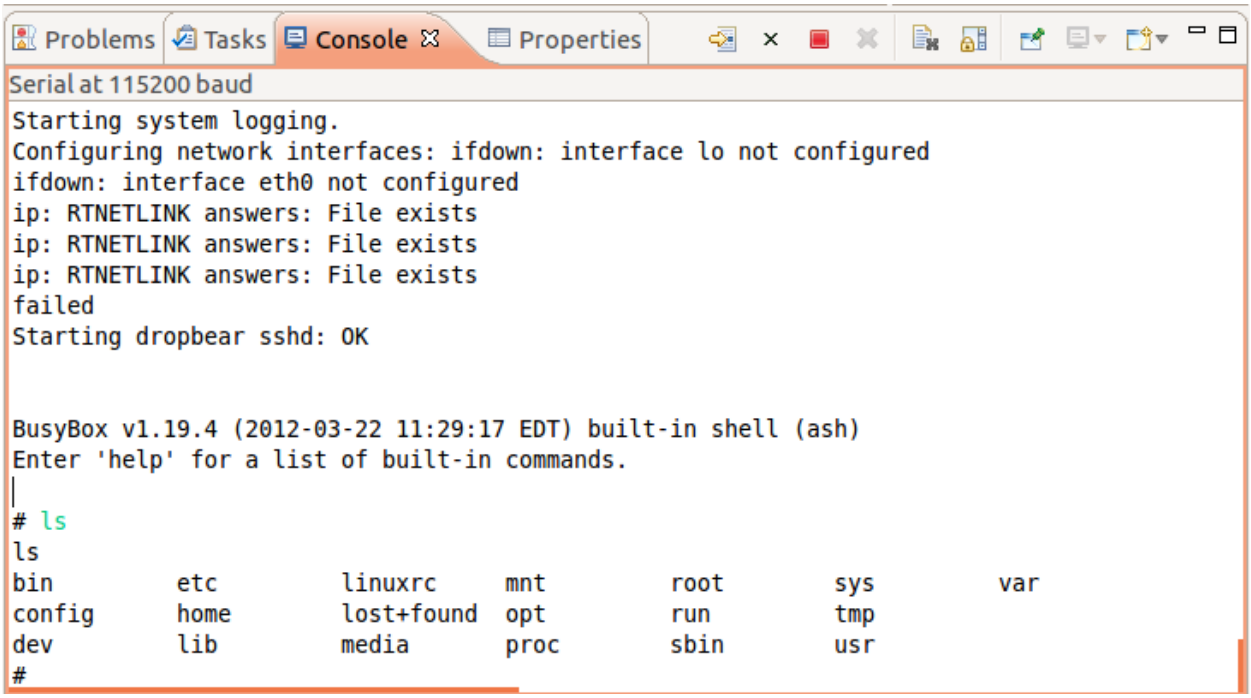


Figure 11: Hardware Target console

The connection used with the target is displayed at the top of the console view. In the console view, you can type commands you want to run on the target and view the output. Please note that this is a basic console view and various control characters and special characters may not be displayed well. This console view will be good for viewing the boot loader output (using a serial connection), run basic commands, like browse file system, change file permissions etc, and copy files to/from the target.

Use the icons on the top right corner of the view to copy the console output to a file, send a ctrl+c command to the target, clear the console and disconnect from the target.

Run/Debug Configurations

TimeStorm IDE uses Run configurations to store download and execution information for your applications. Although Run configurations are part of the basic Eclipse IDE, TimeStorm Run configurations are customized to support executing and debugging a program and its libraries on a remote target.

The C/C++ Development User Guide online help includes additional information about TimeStorm's debugging tools.

Creating a New Run Configuration

To create a new run configuration, select *Run > Run Configurations*… from the main menu. Select the type of configuration that you want in the 'Configurations' list, and click 'New' icon or double-click the 'Configuration type' to add a new configuration. Then, use the tabbed panels on the right to select and configure the run options. Different options appear depending on the type of run configuration that you create.

TimeStorm projects typically use one of the following types of run configurations:

C/C++ Application — Runs a C or C++ project on the local system. Use this option if when running or debugging an application built with the host compiler

TimeStorm C/C++ Remote — Runs or installs a C or C++ project on a remote target. Use this option to debug programs built with this cross-compiler.

Main Panel

In the Main panel *(shown in Figure 12)*, specify the project to run with this *co*nfiguration. You can use the 'Browse' button to search for your project name. After filling in the project name, use the 'Search Project' button to find the executable file that you want to transfer and run.

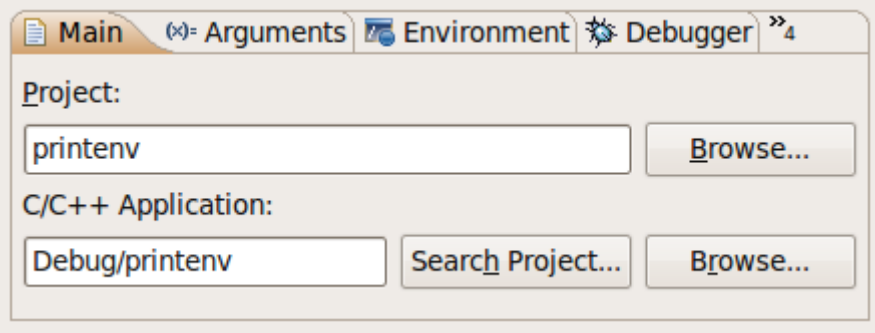


Figure 12: Main Panel

Arguments Panel

In the Arguments panel *(shown in Figure 13)*, you can specify execution details for your application. Only the C/C++ Remote projects have the 'Remote Working Directory' option.

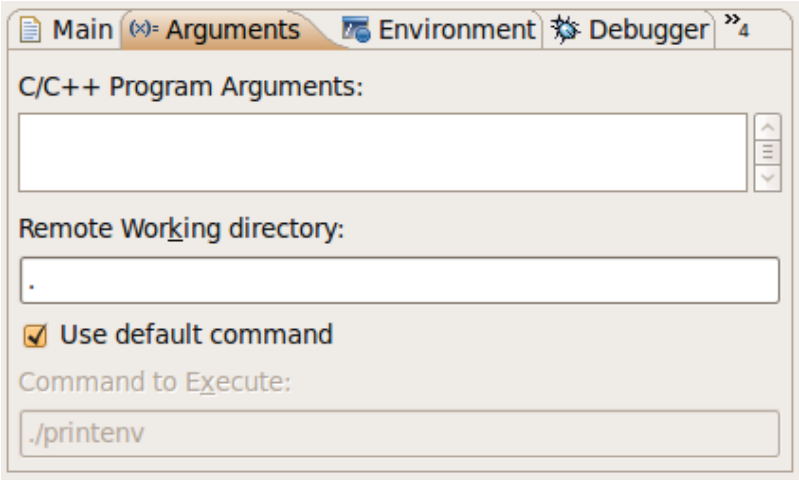


Figure 13: Arguments Panel

C/C++ Program Arguments — You can specify values to pass to the application by typing them in the 'C/C++ Program Arguments' field. Separate values with spaces. Values are passed in the order in which you list them here.

Remote Working Directory — In the Remote 'Working Directory' field, you can specify the directory from which to run the application on the target. TimeStorm switches to this directory path before executing the application.

If you do not change the value of this field from "." the application runs from the target user's home directory.

Command to Execute — In the 'Command to Execute' field, enter the exact command to use when running the application. This value is required. The default value is to run the specified program within the working directory. Uncheck 'Use default command' to change this value.

Keep in mind the directory structure on your target when constructing this command. For example, if you have copied your application file to a directory named `/apps` but you want to execute it from a working directory named `/home/test`, you must include the path to your application in your command.

For example:

```
../../apps/my_app
```

Environment Panel

In the 'Environment' panel *(shown in Figure 14)*, you can set environment variables for the target.

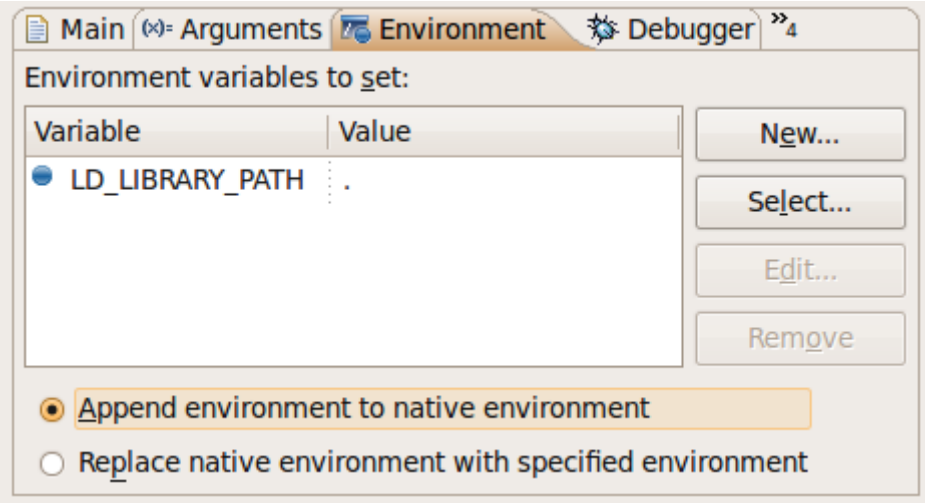


Figure 14: Environment Panel

The panel shown in *Figure 14* contains the following buttons:

New — Click the 'New' button to create a new entry, and the 'New Environment Variable' dialog will appears, as shown in *Figure 15*, below. Click the 'Variables' button to select the variables that you want to use. Then, click 'OK' twice.

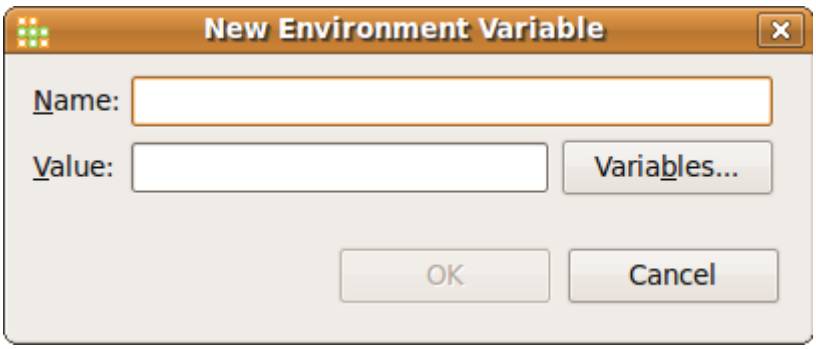


Figure 15: Adding a New Environment Variable

Select — Click the 'Select' button to import environment variables from the host file system.

Edit — To change an entry, select it from the list, and click the 'Edit' button.

Remove — To delete an entry, select it from the list, and click the 'Remove' button.

Debugger Panel

In the Debugger panel, shown in *Figure 16*, you can configure how to debug your project.

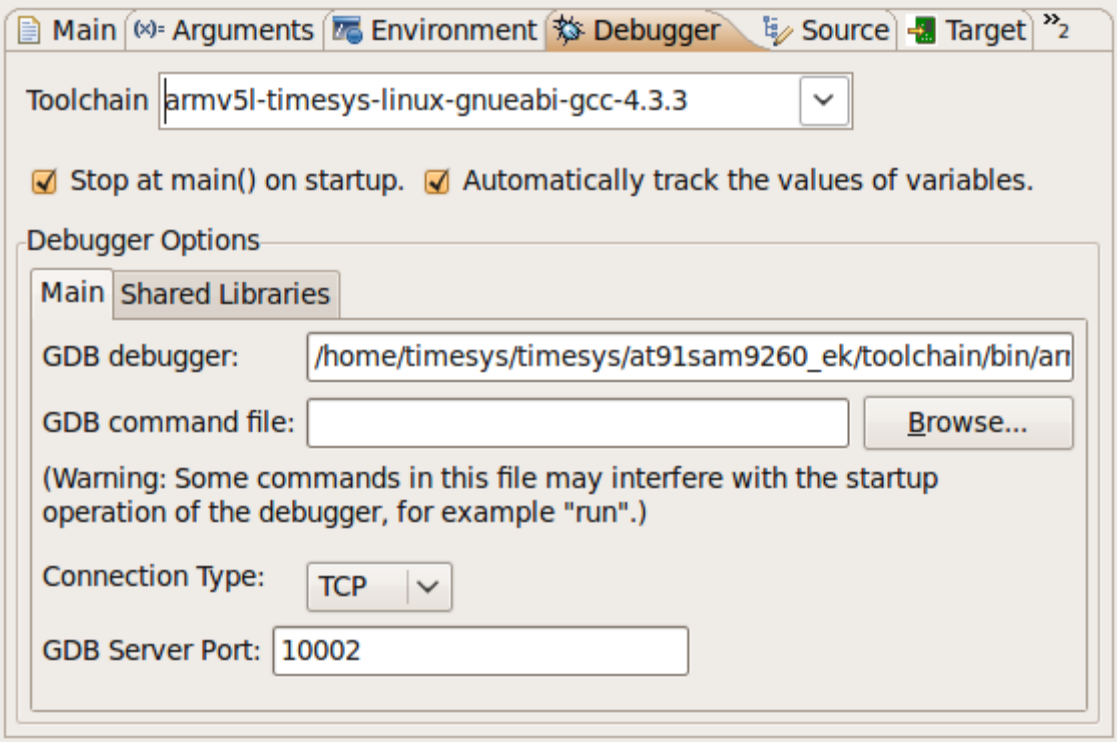


Figure 46: Debugger Panel

This panel contains the following options:

Toolchain — Choose the toolchain to use from the drop-down list of defined toolchains. The path to the debugger executable is set automatically when you choose a toolchain from the list. By default, the toolchain will be the same as the toolchain of the project.

To use a debugger in TimeStorm, you must associate the debugger with a toolchain. You can associate any debugger with a toolchain that you define.

Stop at main() on startup — Select this checkbox if you want execution to pause when your application starts. (This option is selected by default.)

Automatically track the values of variables — Select this checkbox if you want to have the values of variables displayed in the 'Variables' view while your project is being debugged. (This option is selected by default.)

Debugger Options — This section of the panel includes two tabs: 1.) Main and 2.) Shared Libraries. The 'Main' tab is shown in *Figure 16*, above.

Main — This tab includes the following fields:

GDB Debugger — This field automatically displays the default debugger for the toolchain that you have installed and is read-only.

GDB command file — This field allows you to specify a `.gdb-command` file using the Browse button. The debugger will execute the commands specified in this file.

Connection Type — This field allows you to choose whether to use TCP or Serial to establish a debugger connection between host and target. Note that this connection is in addition to the execution-type that you specified in your hardware target. In order to use serial, you must have another serial port in addition to the one being used as the target console.

GDB Server Port — This field lets you specify the port number used for debugging when TCP connection type is selected. This field is visible only when TCP connection type is selected.

Target device — This indicates the name of the serial device on the target to use for the debugger connection. Host and target device must be connected with an appropriate serial cable. This field is only visible when Serial connection type is selected.

Host device — This indicates the name of the serial device on the host (where TimeStorm is running) to use for the debugger connection. Host and target device must be connected with an appropriate serial cable. This field is only visible when Serial connection type is selected.

Serial baud — This indicates the baud rate to use on the serial line for debugger communication. The default rate of 115200 should be used unless you are having serial communication issues. This field is only visible when Serial connection type is selected.

Shared Libraries — This tab is shown in the following *Figure 17*:

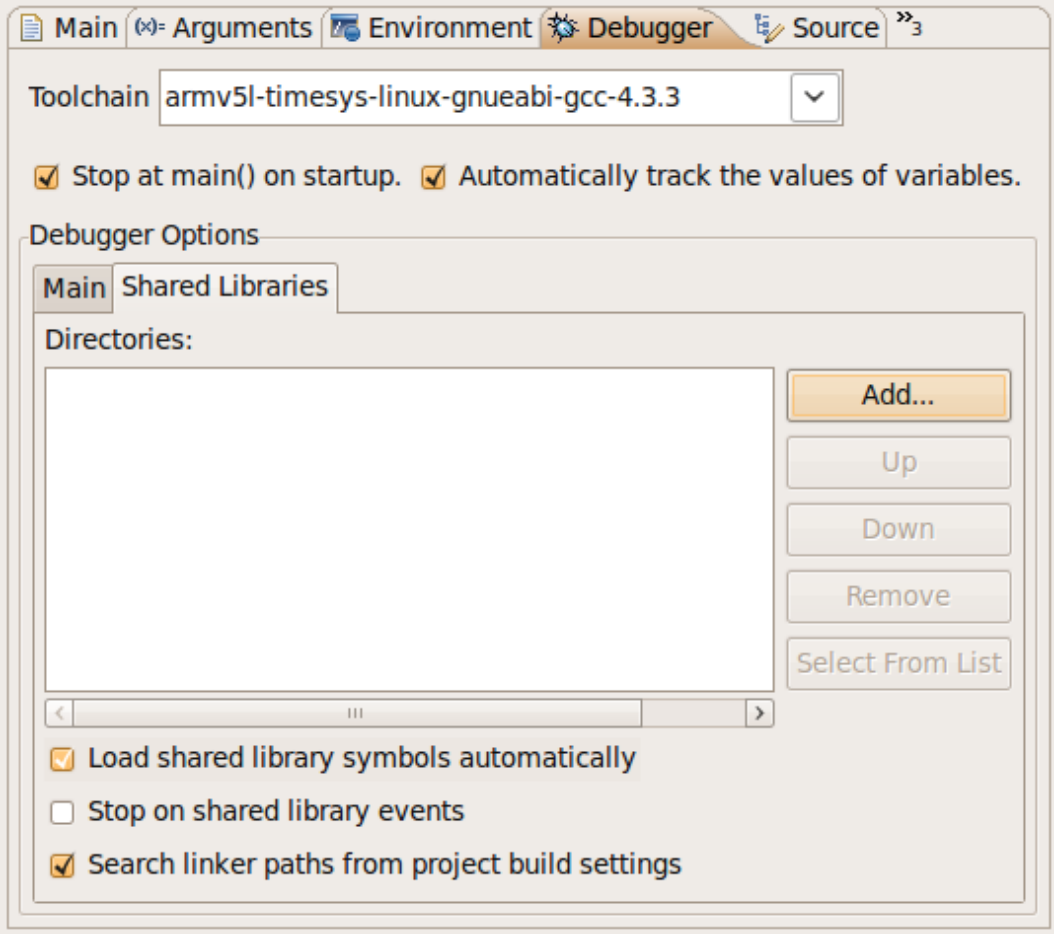


Figure 57: Shared Libraries

The Shared Libraries tab includes the following options:

Directories — This field specifies any additional directories for the debugger to search to find shared libraries (with debugging symbols). By default the debugger will automatically search the paths specified in the project's build settings (see option below). Use the 'Add' button if you want to add other paths. Use the 'Up' and 'Down' buttons to move through the list of directories. Use the 'Remove' button to delete a path.

Load shared library symbols automatically — Select this checkbox if you want these library symbols to be displayed as loaded in the 'Shared Libraries' view and if you want the debugger to hit any breakpoints in the shared library project. (This option is selected by default.)

Stop on shared library events — Select this checkbox if you want the debugger to stop at shared library events, even before it hits breakpoints in the source code. (This option is not selected by default.)

Search linker paths from project build settings — This option is selected by default and will enable the debugger to automatically search all of the linker library path specified in the project's build settings. These will be searched in addition to any paths specified in the Directories field above. To deselect this option, uncheck the box.

Source Panel

The Source panel *(Figure 18)* shows the location of the source files for the project.

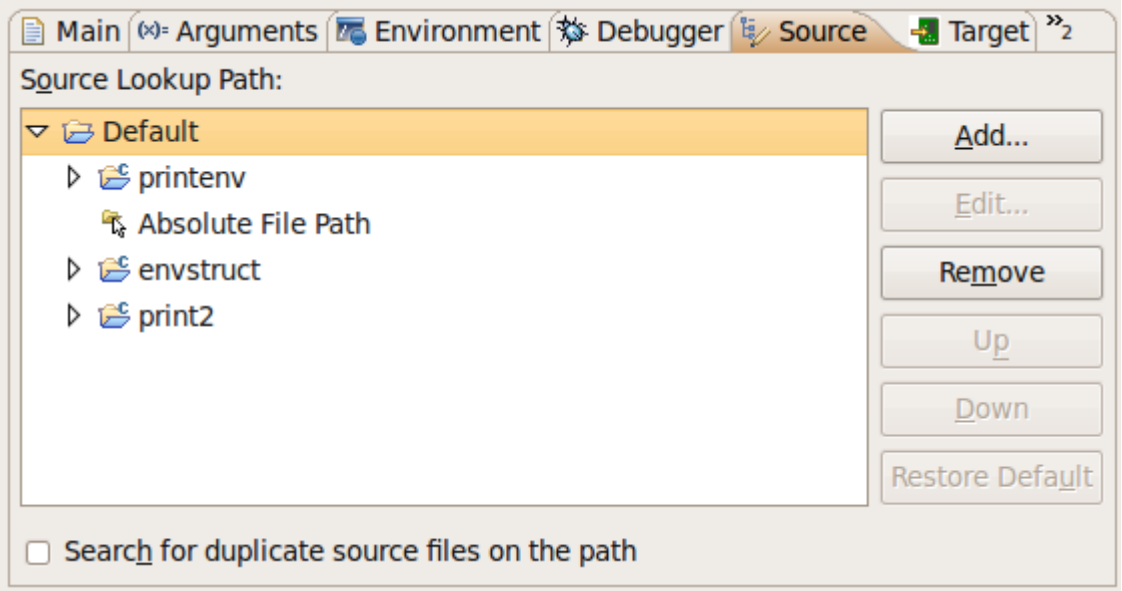


Figure 68: Source Panel

Source Lookup Path — This field shows the project being run, as well as any projects that it references.

Add — You can add arbitrary source file locations by using the Add button. These locations are searched after the generic locations, from the top to the bottom item in order.

Edit, Remove, Up, Down — The other buttons to the right of the list allow you to edit, remove, or reorder the list items, and to restore the default information.

Search for duplicate source files on the path — Selecting this checkbox causes TimeStorm to notify the user if it is unable to determine which source file corresponds to an executing binary file. For example, if you have two source files with the same filename in different directories of the search path, TimeStorm is unable to determine which file is related to the binary file with that name. If this checkbox is selected, TimeStorm displays a message and asks the user to select which file to use. This checkbox is not selected by default.

Target Panel

The Target panel *(shown in Figure 19)*, allows you to select the hardware target used when you run the application.

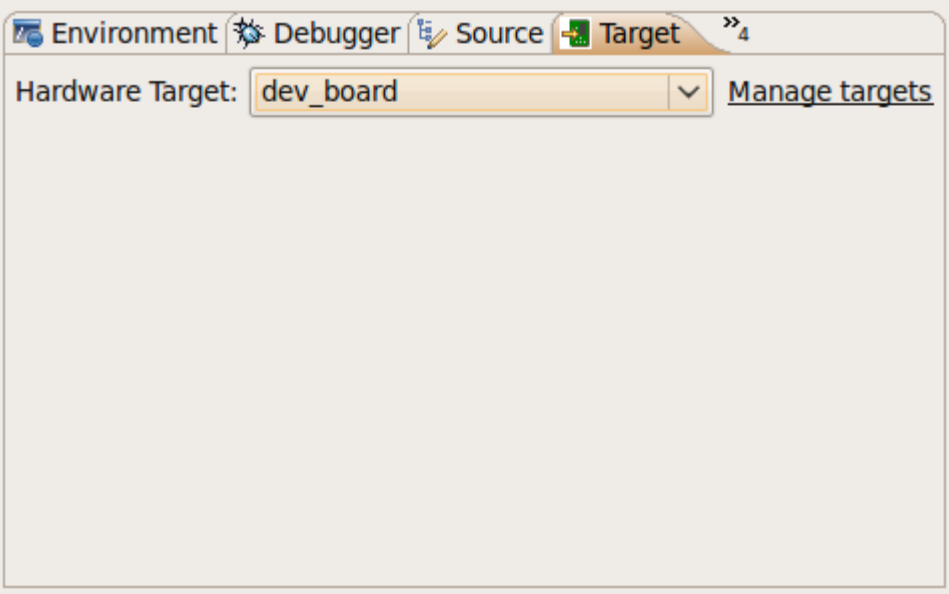


Figure 79: Targets Panel

In the Target panel, use the drop-down menu to select a hardware target that you have registered with TimeStorm. If you have not yet registered a hardware target, click 'Manage targets' to open the Hardware Targets management utility.

If you have problems maintaining a connection to the target when using this run configuration, consider increasing the value in the timeout setting.

Settings made in the 'Launch Timeout' panel only affect run configurations; they do not affect timeout settings for the 'Console' view or for any other use of Telnet or FTP.

Telnet and FTP timeout values for run configurations can be set from the TimeStorm Preferences panel. From the main TimeStorm menu, select *Window > Preferences* and open the *TimeStorm > Launch Timeout* item.

Download Files Panel

In the 'Download Files' panel (*shown in Figure 20)*, you can select the files that TimeStorm transfers to your target. You can use this panel to download additional files to your target, along with your application.

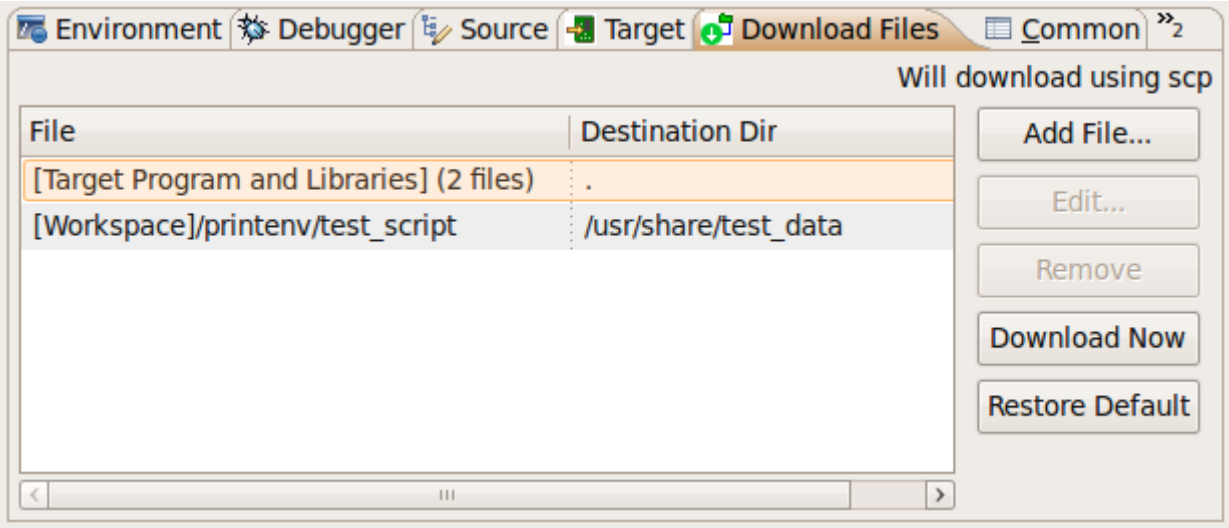


Figure 20: Download Files Panel

The application specified in the File field is listed as `[Target Program]`. This is the application that will be run. You can change where it is downloaded by selecting it and clicking Edit.

If the program links in shared libraries, then these libraries will be downloaded to the target as well and the Files field is listed as `[Target Program and Libraries]`. These libraries will be downloaded to the same location as the target program. To download the program and libraries to separate locations, remove the default entry and re-add each as a separate entry.

File paths are interpreted as relative to the target directory to which you download files. If you transfer files by using FTP, the actual destination directories depend on the configuration of your FTP software. In some cases, directories specified in this panel are appended to the user's home directory on the target.

This panel contains the following buttons:

Add File – Use the 'Add File' button to specify additional files to transfer, along with the application. These files are transferred every time you transfer the application file.

Edit – To change where the application (or any other file in this list) is installed, select it and click 'Edit' to change its destination directory.

Remove – To eliminate a file from the items to download, select it in the list and click the 'Remove' button.

Download Now – The 'Download Now' button copies files immediately, without running the application.

Restore Default – The 'Restore Default' button resets the panel to its initial state (download only the application and its libraries).

> ***NOTE:*** *Downloaded files overwrite any identically named files on the target without giving a warning. TimeStorm will not create destination directories on the target, so be sure to create them before launching the configuration.*

Common Panel

The 'Common' panel *(shown in Figure 21)*, sets options for sharing this run configuration among multiple projects.

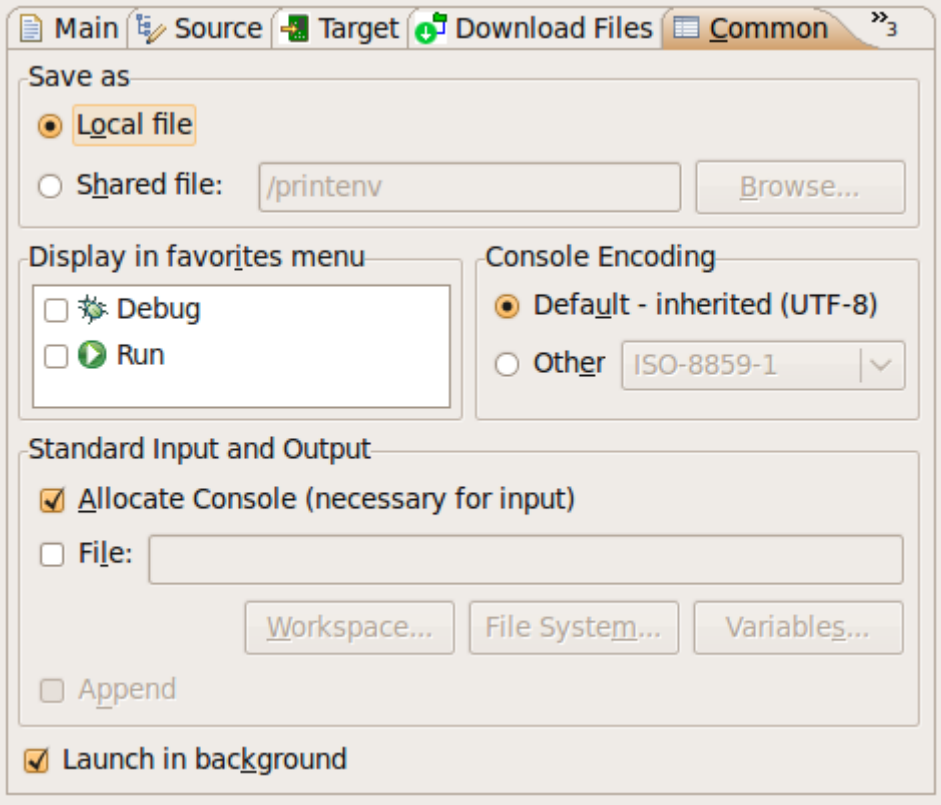


Figure 21: Common Panel

The options with the Common Panel include:

Local File — By default, run configurations are saved with the workspace state files, so they can only be used with projects in the current workspace. (This setting corresponds to the 'Local file' radio button on this panel.) However, you can make the configuration available for use in other workspaces by choosing the 'Shared file' radio button.

Shared File — When you select this option, the run configuration is saved as a `.launch` file that can be imported into another TimeStorm workspace. Optionally, you can specify a different location for the file so that it is more easily accessible to multiple projects.

Display in Favorites Menu — You can select whether to include this run configuration on the main 'Run' and 'Debug' menus. For example, if you select the 'Run' checkbox, the run configuration always appears in the 'Run History' submenu.

You can choose to include the run configuration in either menu, both menus, or neither menu.

Launch in Background — Selecting the 'Launch in background' checkbox causes this run configuration to run as a background thread. Running as a background thread is the default value. Clear this checkbox if you want to run in the foreground, along with other TimeStorm processing.

NOTE: *The other options in this panel are Eclipse features that are not currently supported in TimeStorm.*

Creating an Application Project

Application projects are created by using the New C or C++ Project wizard. TimeStorm uses the existing CDT C and C++ Project wizards, but adds the ability to select a cross-toolchain. By creating a project with cross-toolchain, TimeStorm will be able to build the project with one or more of the toolchains placed in the system when installing a Timesys starting point.

Create New Project

To create a new application project:

Open the 'New Project' wizard by choosing *File > New > Project* from the main menu. *OR* Click the 'New' button  and use the drop-down menu to the right of the button to choose 'Project'.

1. Choose a C or C++ Project. The new C/C++ Wizard *(shown in Figure 22)* can be used to create several different types of projects, including 'Executable' and 'Shared and Static Library'. This section will focus on creating an Executable project. Shared and static library projects are covered in the Creating Shared and Static Libraries section of this manual.

Project Panel

1. Under the 'Project Type' heading, expand the Executable type and select the source code template you would like to use.

2. Under the Toolchains heading select the TimeStorm Cross-Compile Toolchain.

3. Click 'Next' to bring up the 'Select Configurations' panel *(shown in Figure 23)*.

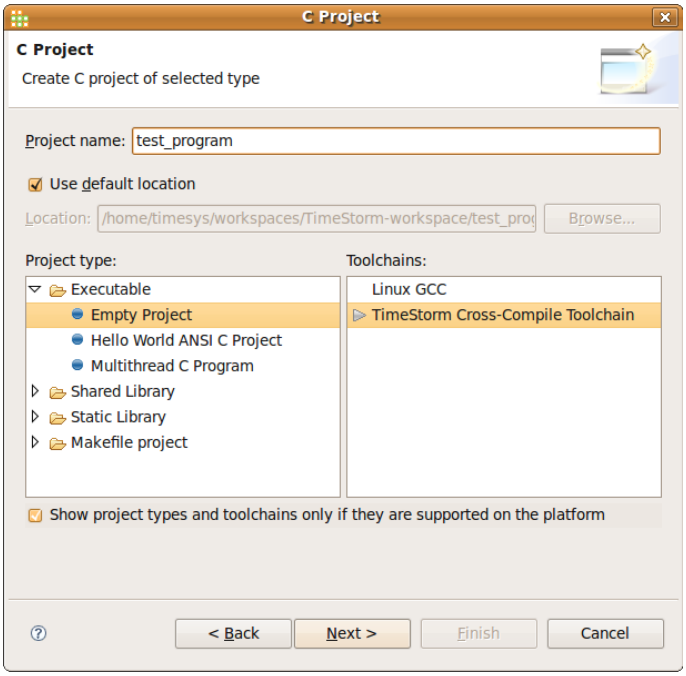


Figure 22: New C/C++ Project Wizard

Select Build Configuration Panel

In this panel, TimeStorm will suggest creating three build configurations that contain the compilation settings for the project. Build configurations work together with the toolchain and project to create the makefile that TimeStorm uses for the build. For more information on Build Configurations, please see the How TimeStorm Builds a Project section.

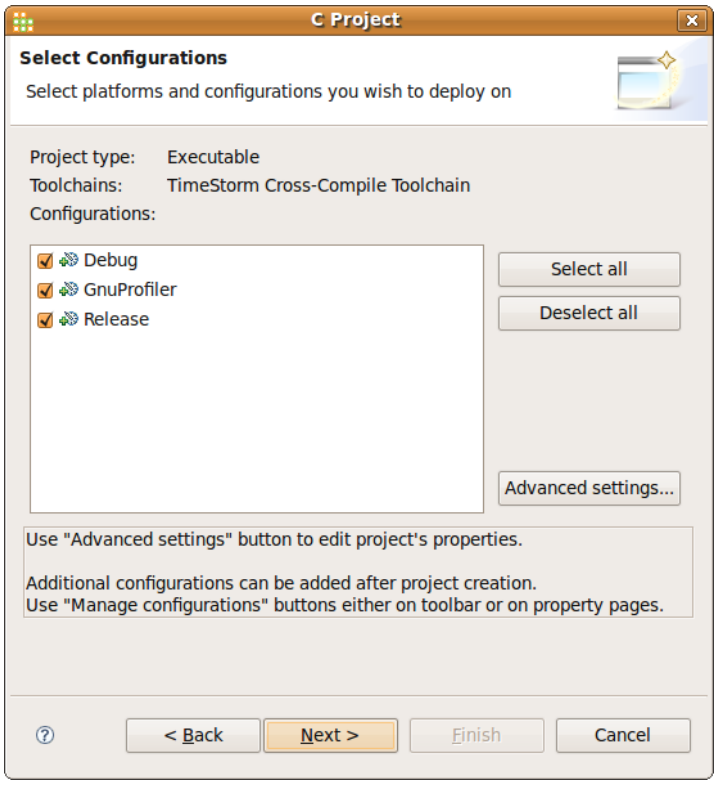


Figure 23: Select Build Configuration panel

By default, TimeStorm creates three Build Configurations for new projects:

Debug — This setting performs no code optimization and attaches complete debugging information. These settings are designed make debugging as easy as possible.

Release — This setting has the highest compiler code optimization settings. The output will not include any debugging symbols. Code compiled with this configuration is ready for production use.

GnuProfiler — This setting builds the software with debugging information as well as code to collect profiling information via GNU gprof.

Select a Toolchain Panel

The next panel in the wizard *(shown in figure 24)* asks the user to select a toolchain. The entries in this list appear because the user has installed a Timesys Starting Point or have added the toolchain manually. Please refer to the Working with Toolchains section to understand how TimeStorm manages toolchains. The toolchain can be changed later within the project's property page.

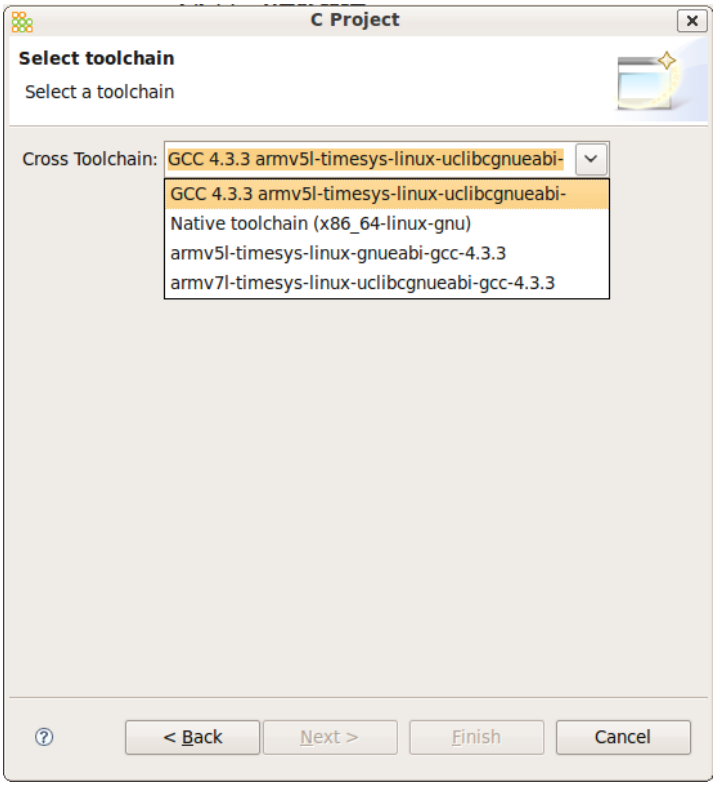


Figure 24: Select Cross Toolchain

Once the 'Finish' button is clicked, the project will be created in the Workspace.

Creating Static and Shared Libraries

Library projects are created by using the 'New C or C++ Project' wizard, shown in *Figure 25*. TimeStorm uses the existing CDT C and C++ Project wizards, but adds the ability to select a cross-toolchain. By creating a project with cross-toolchain, TimeStorm will be able to build the project with one or more of the toolchains placed on the system when installing a LinuxLink.

Open the New Project wizard by choosing *File > New > Project* from the main menu or by clicking the 'New' button  . Optionally, use the drop-down menu to the right of the button to choose 'Project.' Choose a C or C++ Project.

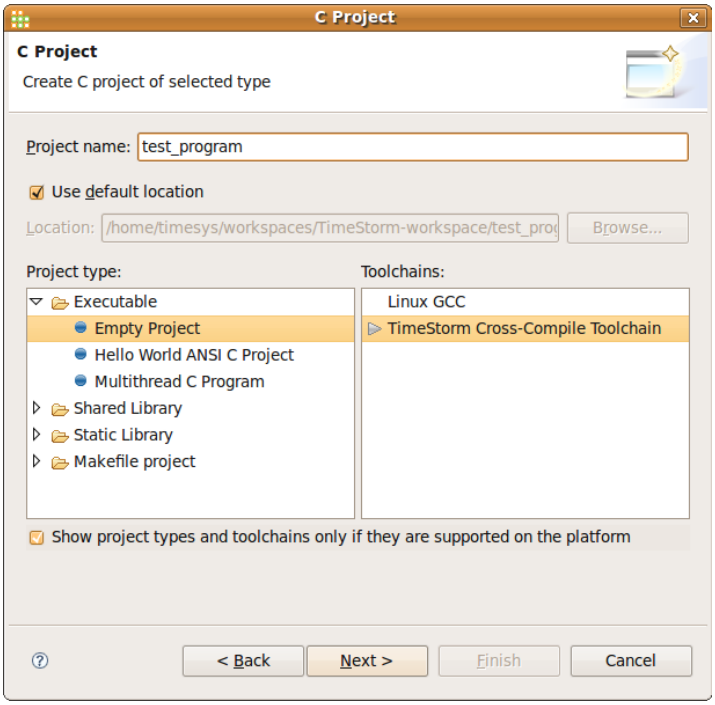


Figure 25: New C/C++ Project Wizard, Creating Libraries

Static versus Shared Libraries

Static and shared libraries allow the user to place certain functionality outside of the main program, frequently so the same library can be shared across projects. Static libraries are incorporated by copying the bits into the main program at link time while shared libraries are linked into the program before it runs by a dynamic library loader. Shared libraries can also be accessed when the program is running by loading the libraries into memory and calling functions, without the linking step before the program runs.

	Library Type	
Feature	Static	Shared
Output File Name	lib<project name>.a	Lib<project name>.so
Can Update at Run-time	No	Yes
Dynamic Loader Needed	No	Yes
Code Shared Across Applications	No	Yes

Select the TimeStorm Cross-Compile Toolchain from the list of toolchains populated, installed from a Timesys Starting Point or added manually to TimeStorm. The toolchain can be changed later in the Project's Property page (see Changing the Toolchain for an Existing Project). Next select the configurations: Debug, Profile, and Release. After clicking the 'Finish' button, the project will be created in the Workspace.

Editing and Building

After starting development using the wizard, TimeStorm works like a standard IDE. When new files are added to the project, TimeStorm will add them to the project's make file and will use the program's extension to determine what tool to use for the build process.

Extension	Build Program
.c, .C	C compiler
.c++, .cxx, .cpp, .cc	C++ compiler
.s	Assembler

To add an include file, source file, or create a sub directory to a project, right click on the project, choose 'New' to see different file options. Give the appropriate file extension when specifying the file name.

To build the application, select *Project > Build Project* from the menu. TimeStorm will build the project with the active build configuration and display any errors or warnings in the problems view. During the build process, TimeStorm generates a make file for the project based on the active build configuration as described below in the How TimeStorm Builds a Project section of the manual. A project created by a TimeStorm wizard is designed to build without any errors or warnings.

How TimeStorm Builds a Project

TimeStorm uses Build Configurations to control the project build process. When creating a project using a wizard, several build configurations will be created by TimeStorm and associated with the project. One of these build configurations is the "active" configuration (as illustrated by the star in the illustration, below) and that will be used by default when creating a build. Any build configuration can be selected as the active configuration, as this concept exists so the user does not need to select which build settings to use when creating a build.

Figure 26, below, shows the relationship between the project, build configurations, toolchains and the make files generated by TimeStorm.

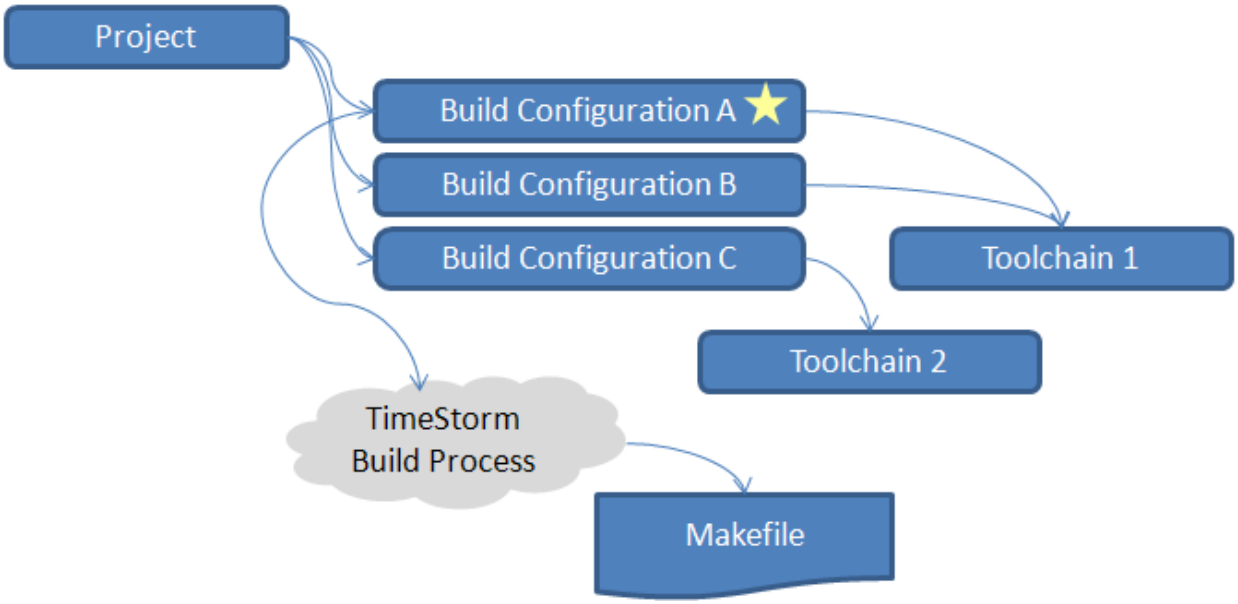


Figure 26: Relationship Between Project, Build Configuration, Toolchain and Makefile

The following describes how TimeStorm uses the makefile and build config to build the project. TimeStorm will use the default build configuration, following these steps:

1. **Create a directory with the name of the build configuration** — The output for the build is stored under a directory that matches the name of the build configuration used for the build. If this directory does not exist, it will be created. TimeStorm must have the ability to create directories in the workspace or this operation will result in an error.

2. **Scan the project, create a makefile that invokes the appropriate build program for the source file** — Before the build process occurs, TimeStorm will scan the project for files it knows how to build based on extension, ignoring files that it does not know how to handle. For those files that it knows how to process, TimeStorm will then examine the files, calculate dependencies, and emit a standard GNU makefile.

3. **Execute the makefile, display results in the Console Window** — After creating the makefile, TimeStorm uses GNU to perform the build. The results of the build process are displayed in the Console window.

4. **Scan the results, create markers for errors and warnings** — After the build, the results are scanned and TimeStorm translates errors and warnings for files into markers that appear next to the offending line and in the problems view.

Changing the Toolchain for an Existing Project

If you want to switch to a different cross toolchain or if you want to check if your code compiles with a native toolchain, you can change the toolchain used by a project after creating it.

To change the toolchain *(shown in Figure 27)*:

1. Right-click on the project, and then click 'Properties'

2. In the 'Properties' window, expand C/C++ build and click 'Settings.' The project's build settings is displayed in the right hand side.

3. Click the 'Cross toolchain' tab, and select the toolchain you want to use with the project.
NOTE: *The toolchain you are changing is for the build configuration displayed at the top of the page.*

4. Make sure to 'Clean' and 'Build' the project after you change the toolchain.

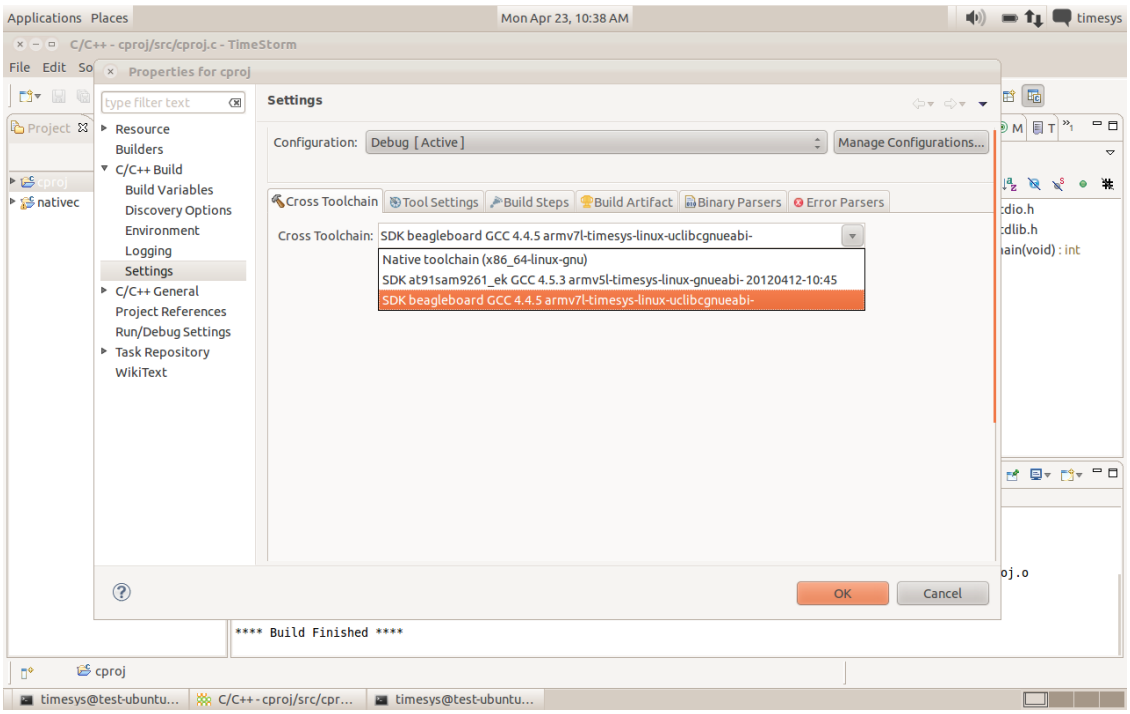


Figure 27: Changing the Toolchain for an Existing Project

Remote versus Local Application Debugging

Application debugging can occur on the host machine or the target board, depending on how the application was built. When debugging on the local machine, the program needs to be built with the host's compiler.

For most projects, it makes sense for debugging to happen on the host machine, as debugging remotely is slower due to the communications link. In addition, most code problems are algorithmic in nature, meaning that the code itself contains problems, independent of where the code is running. This means developers can be very productive by debugging the code on their host machine and then recompiling it for final testing on the target. Some things, like timing, performance or access to specialized devices, cannot be tested on the host machine and in this case remote debugging is the right tool to use from the start.

Debugging on the Development Host

When debugging on the development host, select *Run > Debug Configurations…* from the main menu, and use the C/C++ Application configuration from the debug run configuration dialog.
This run configuration shares many of the same panels as the C/C++ Remote Run configuration and works in much the same way. Just like with the C/C++ Remote Run configuration, the user needs to specify what project and application will be used for debugging and has control over the environment variables and source code locations.

Development Host Debugging Strategies

Since it may not be possible to create a complete environment for running a program on the host machine, the recommended strategy is to build "scaffolding" code to emulate the target device and create an environment similar to the target. To minimize the development efforts for this scaffolding code, engineers will not put effort into creating a high-fidelity emulation, but one good enough to exercise the code in question.

In addition, the host machine is frequently configured to be similar in file structure as the remote machine or a directory is created to resemble the root file system of the target and the program performs a chroot() when starting. Following this strategy allows the developer to create a separate environment that is closer to that of the target, that is with the same set of files, permissions, libraries and device nodes without interfering with the host machine's configuration. The chroot() strategy also helps reduce regressions that can happen when recompiling and testing the program on the target machine.

Building Related Projects in a Workspace

In TimeStorm, projects are not hierarchical. This is much different than other systems where a project is frequently structured as a top-level directory with a directory for each component, with those directories nesting downward.

For example:

```
top-level-dir/
      application
      shared-lib-1/
            static-lib-1/
            static-lib-2/
      shared-lib-2/
            static-lib-1/
            static-lib-2/
```

In this project, the user would typically write a make file that built the projects in the following order:

```
shared-lib-2/static-lib-2/
shared-lib-2/static-lib-1/
shared-lib-1/static-lib-2/
shared-lib-1/static-lib-1/
application/
```

In TimeStorm, since all of the projects are peers of each other, enforcing the build order would occur through 'Project References', which is part of the project properties. To access the Project References:

1. Open the 'Project Properties' dialog by right-clicking and selecting 'Properties.'
2. Select the 'Project References' entry on the left. The dialog will look like the following, shown in *Figure 28.*

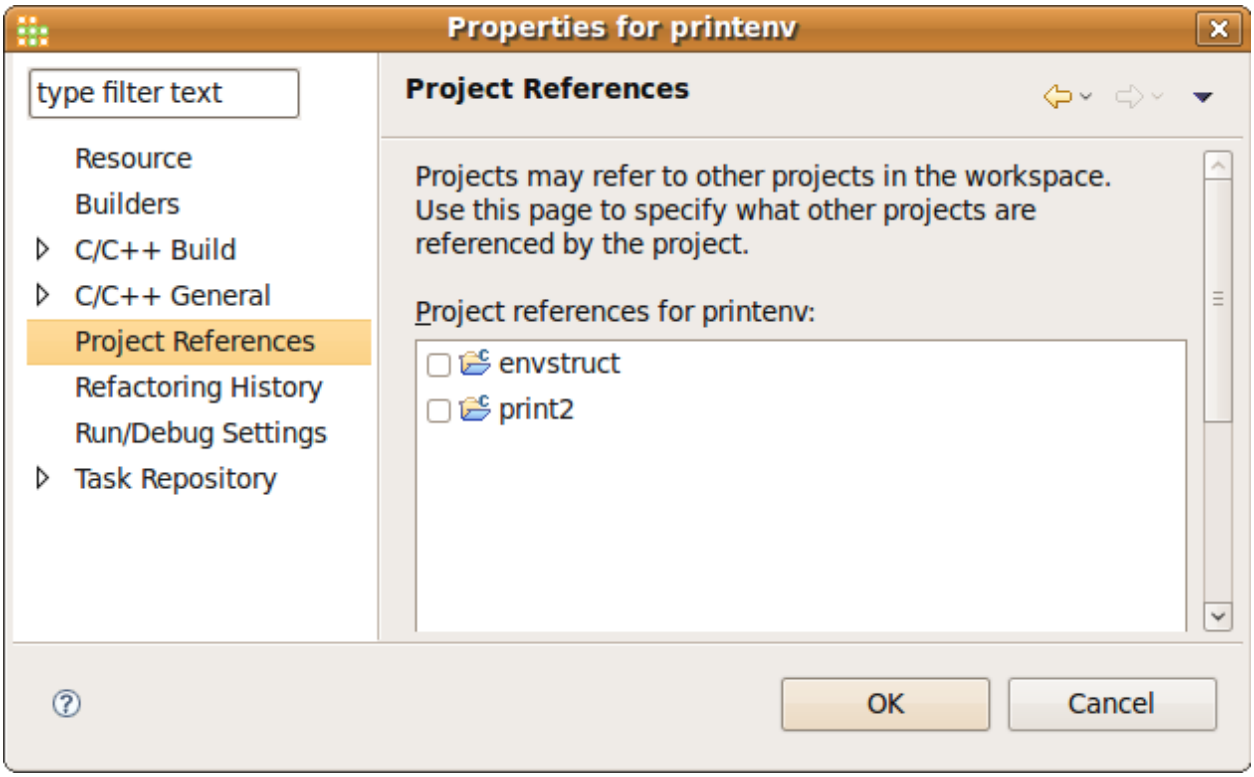


Figure 28: Project References Panel

By selecting entries in this dialog, TimeStorm will build those projects before the current project (if, in fact, they needed to be rebuilt). Project references can be nested several levels deep, so that if a referenced project has other references, those references also will be rebuilt, if necessary. This ensures that the top-level project has all of its dependencies built before starting its build. In this way, the developer has the same degree of control as one would have when using a traditional "nested" make file.

Integrating application project into Desktop Factory

After you have developed your application in TimeStorm, you can integrate the project into desktop factory that is used by your company's build system. The project will be built using the toolchain that is used by desktop factory.

To integrate an application into desktop factory, build the project in TimeStorm, right click on the project and click Export > C/C++ > Desktop Factory Integration and click Next. This will open a desktop factory integration dialog as shown below in *Figure 29.*

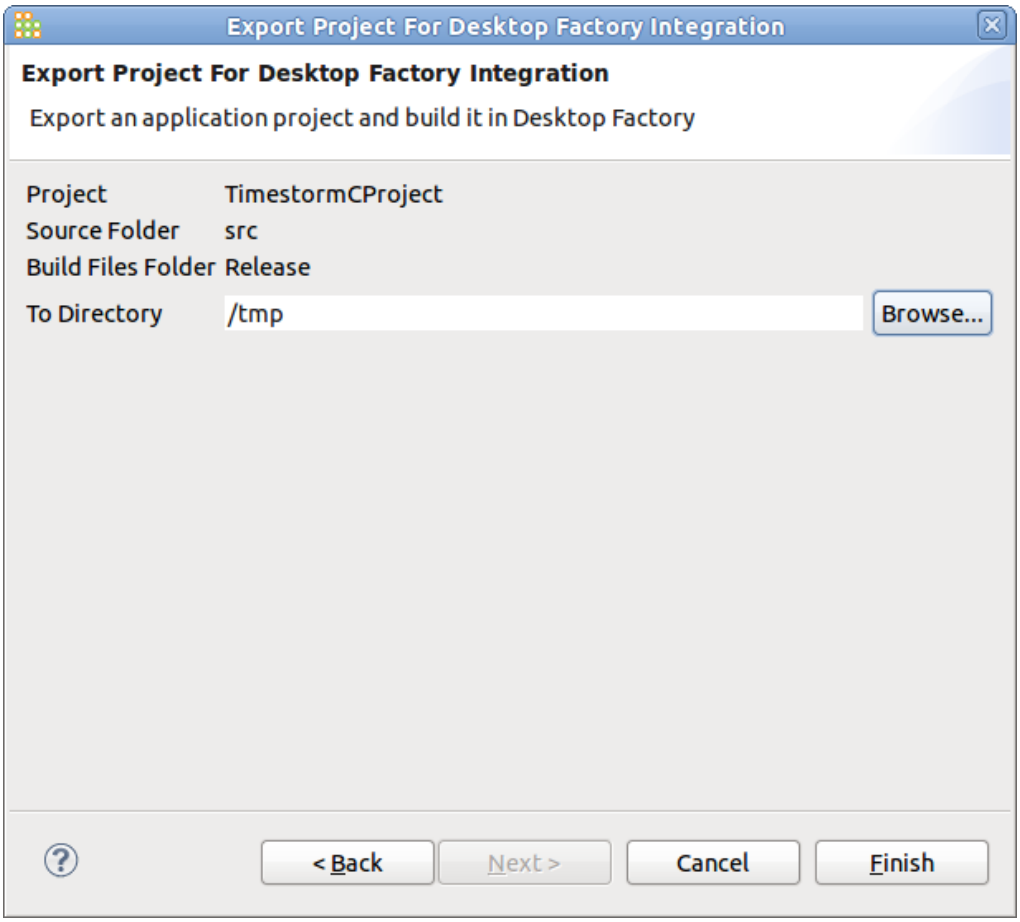


Figure 29: Desktop Factory Integration

Select the directory to copy the exported project tar file and click Finish.

You can share the exported tar file with your build engineer / platform developer to integrate into Desktop Factory. They have to run

```
./bin/install_timestorm_project <project.tar.gz>
```

to integrate the project to desktop factory. For qt projects, they have to append `qt` to the above command. For more details, refer to the desktop factory documentation.

Kernel Development with TimeStorm

Kernel Development includes:

- creating a kernel project
- configuring the kernel
- building and deploying the kernel
- debugging the kernel

Creating a Kernel Project

New Project Wizard

To create a kernel project, open the New Project wizard by clicking File > New > Project from main menu or Click the 'New' button  . Select Kernel > Linux Kernel Project and click 'Next' to continue, as shown in *Figure 30.*

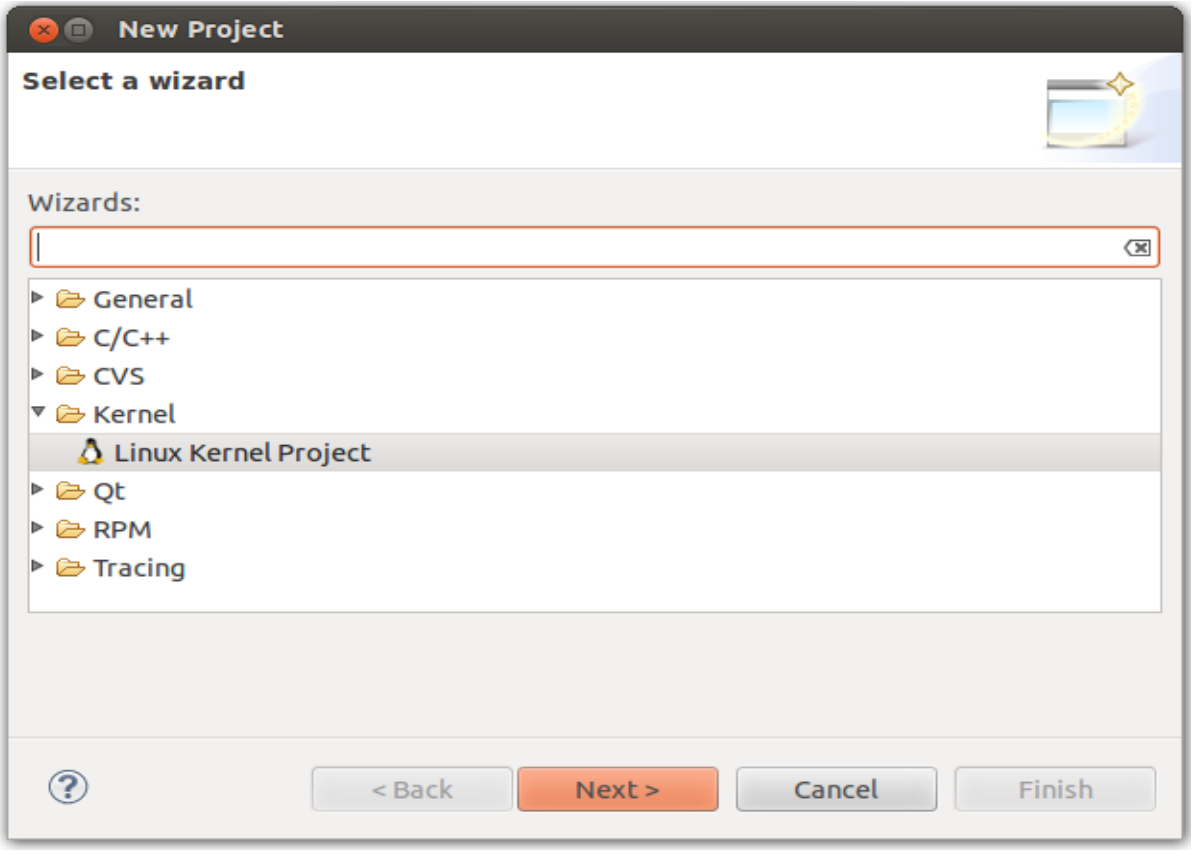


Figure 30: New Project wizard – TimeStorm Linux Kernel Project

Project Page

The page shown in *Figure 31* allows you to enter a name for the the kernel project, and select the kernel sources to use with the project.

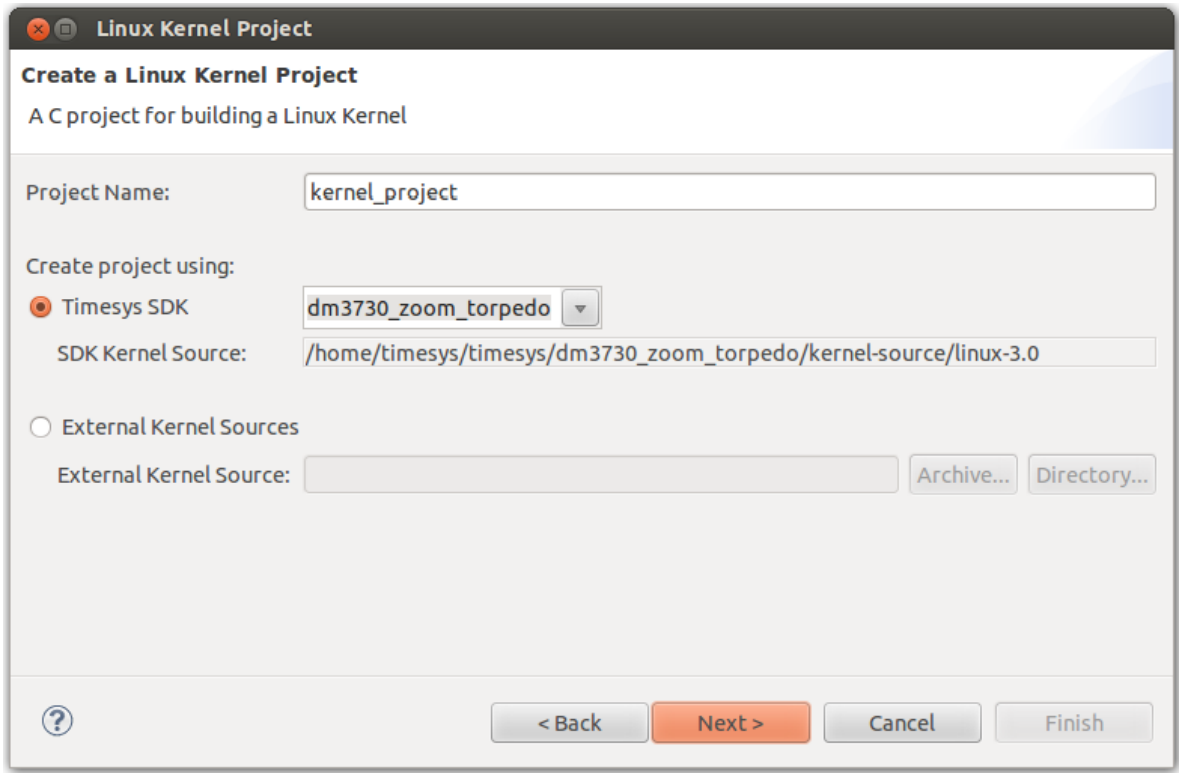


Figure 31: Kernel Project Page

You can create a kernel project using the kernel sources included in the Timesys SDK, or kernel sources you have in an archive file or directory. When you select the SDK, the architecture, toolchain and other details required for building the kernel project are filled in for you on the next pages.

Kernel Source Management

You can create a kernel project by making a copy of your existing kernel sources or use your kernel sources. This page allows you to choose how you want to manage your kernel sources. You can create the project in your kernel source location or at a different location. You can also choose to create the project in TimeStorm workspace or at a different location as shown in *Figure 32*.

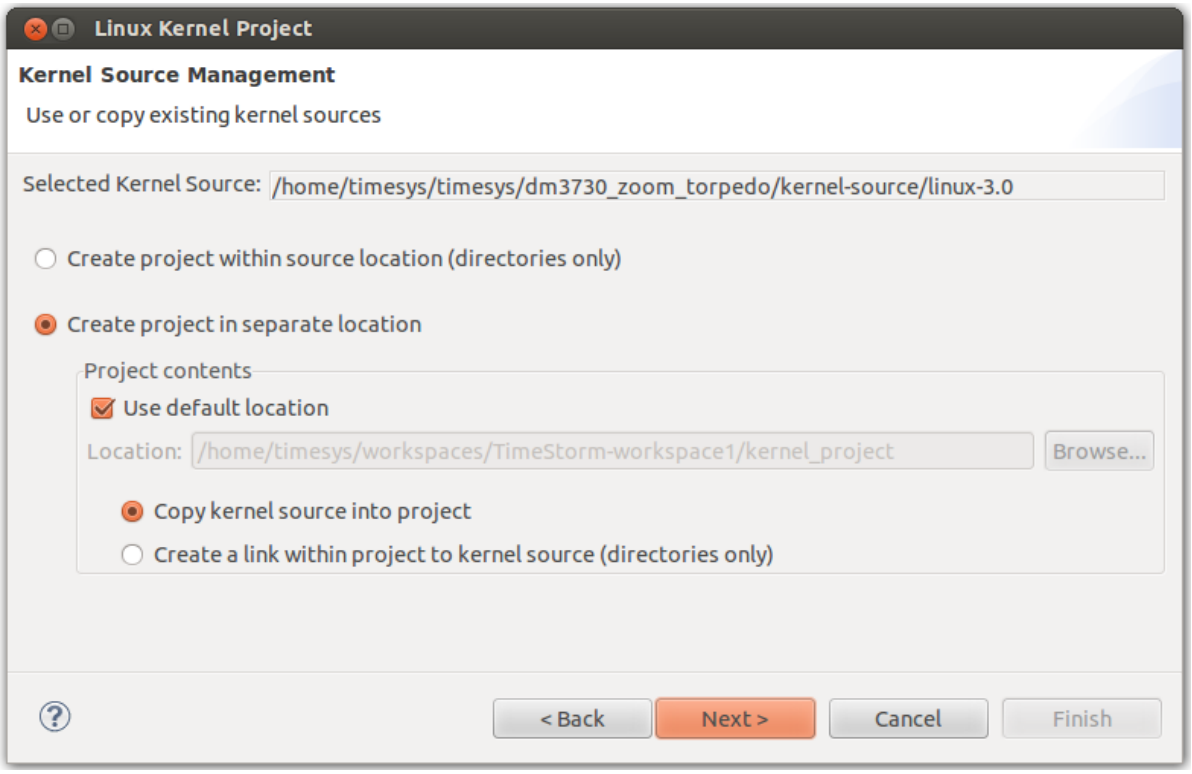


Figure 32: Kernel source management

Kernel Build Settings

Kernel build settings page show in *Figure 33* allows you to select the toolchain, enter the architecture and make target. If you selected the Timesys SDK on the previous page, these values are filled based on the SDK selection. You can choose whether to build the kernel modules along with the kernel.

The cross-compile prefix for the kernel build is set based on the selected toolchain. If you would like to enter a custom cross-compile prefix, you can select None in the toolchain dropdown.

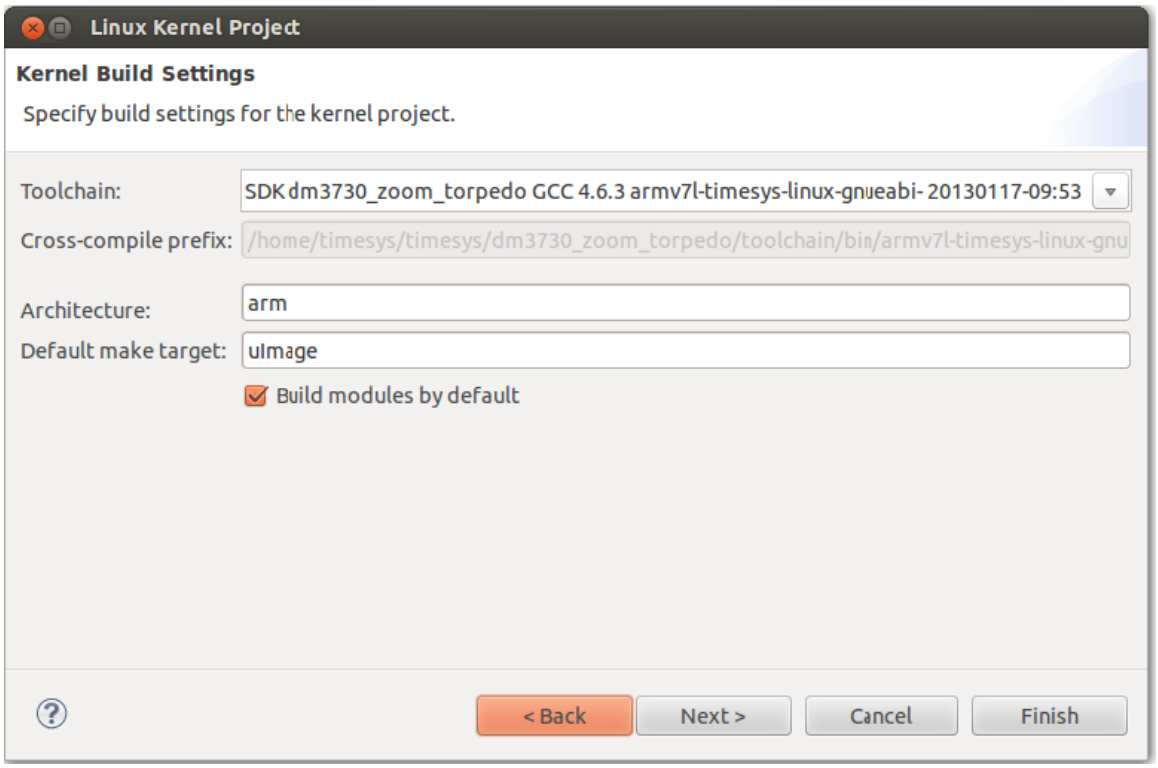


Figure 33: Kernel Build Settings

Build Output

Figure 34 allows you to manage the kernel build output. You can save the kernel build output in a sub-directory in kernel project or an external directory.

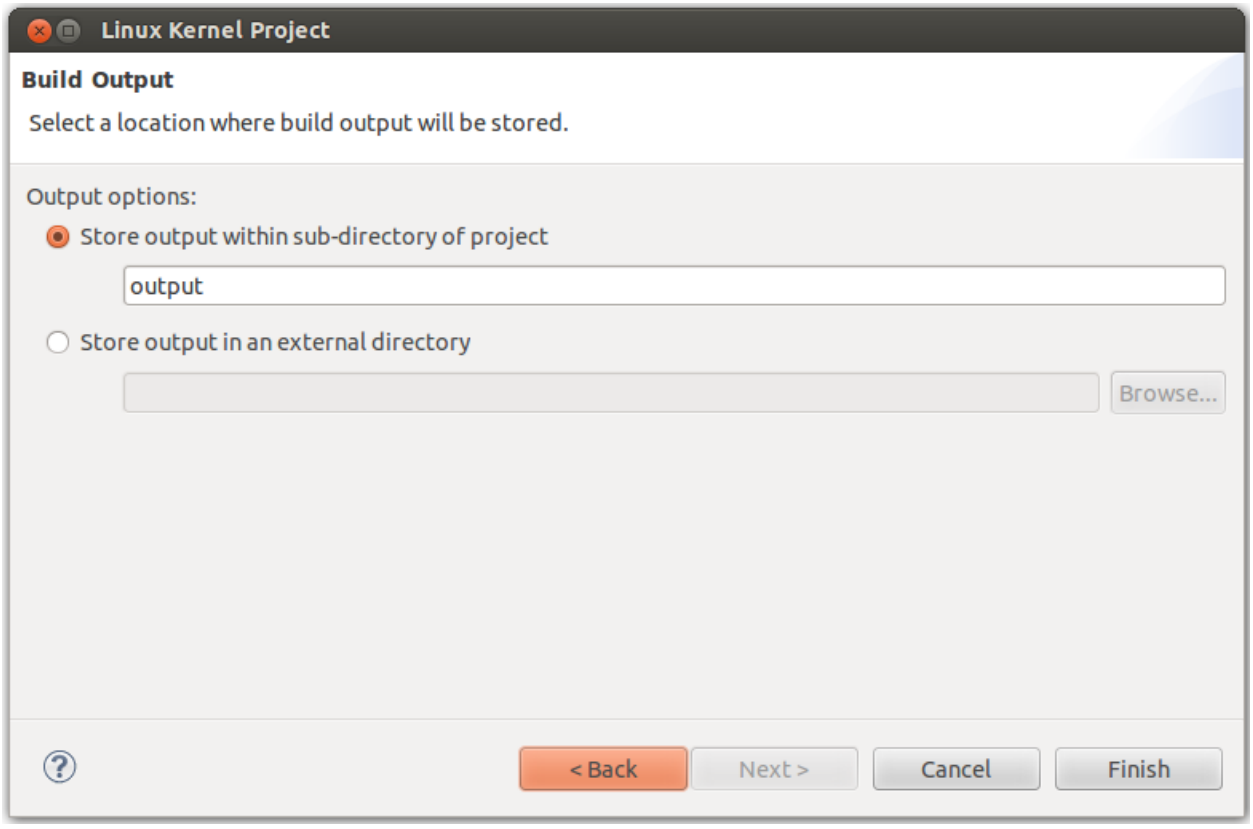


Figure 34: Kernel Build Output

Click Finish to create the kernel project. Copying the kernel sources takes some time and you may notice a delay in creating the project.

Please note that to conserve memory and CPU resources, indexing is turned off for kernel projects. The side effect of this is you may see errors / warnings when you open kernel sources in an editor.

Configuring the Kernel

You can configure the kernel by right clicking on the kernel project, clicking Configure Kernel and then clicking MenuConfig or XConfig. TimeStorm launches the menuconfig or xconfig editor outside TimeStorm. You can configure the kernel and save your changes. Please note that for running menuconfig, you should have libncurses5-dev and xterm package installed, and for running xconfig, you should have qt3-dev-tools or qt4-dev-tools package and xterm installed.

Menuconfig and xconfig editors are shown in *Figure 35* and *Figure 36* respectively.

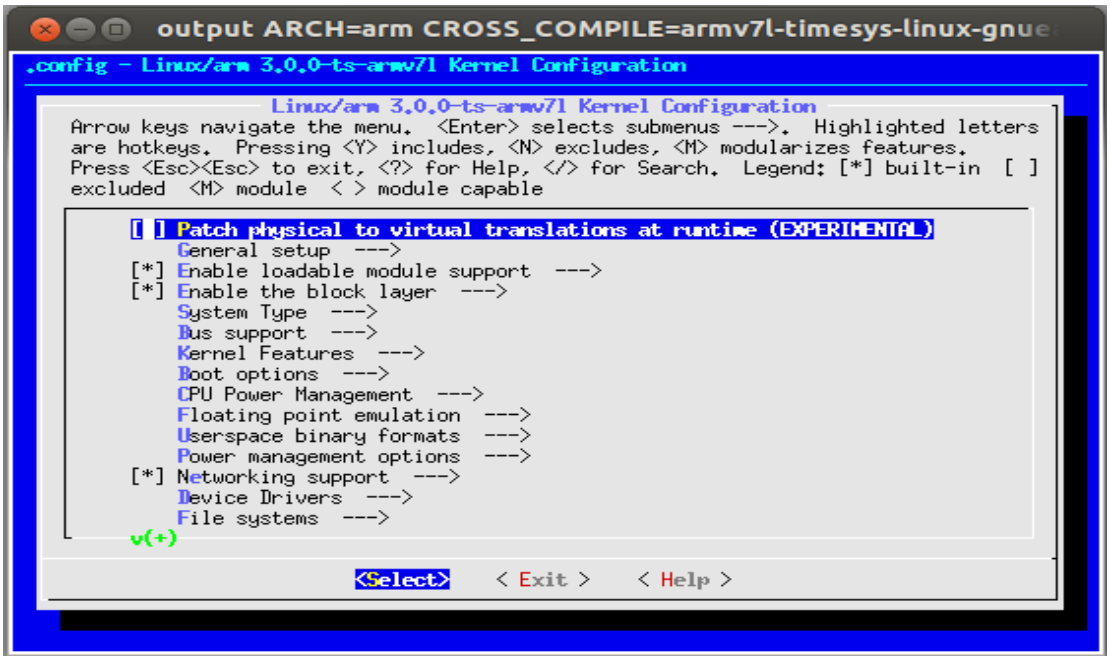


Figure 35: Menuconfig – Kernel Configuration

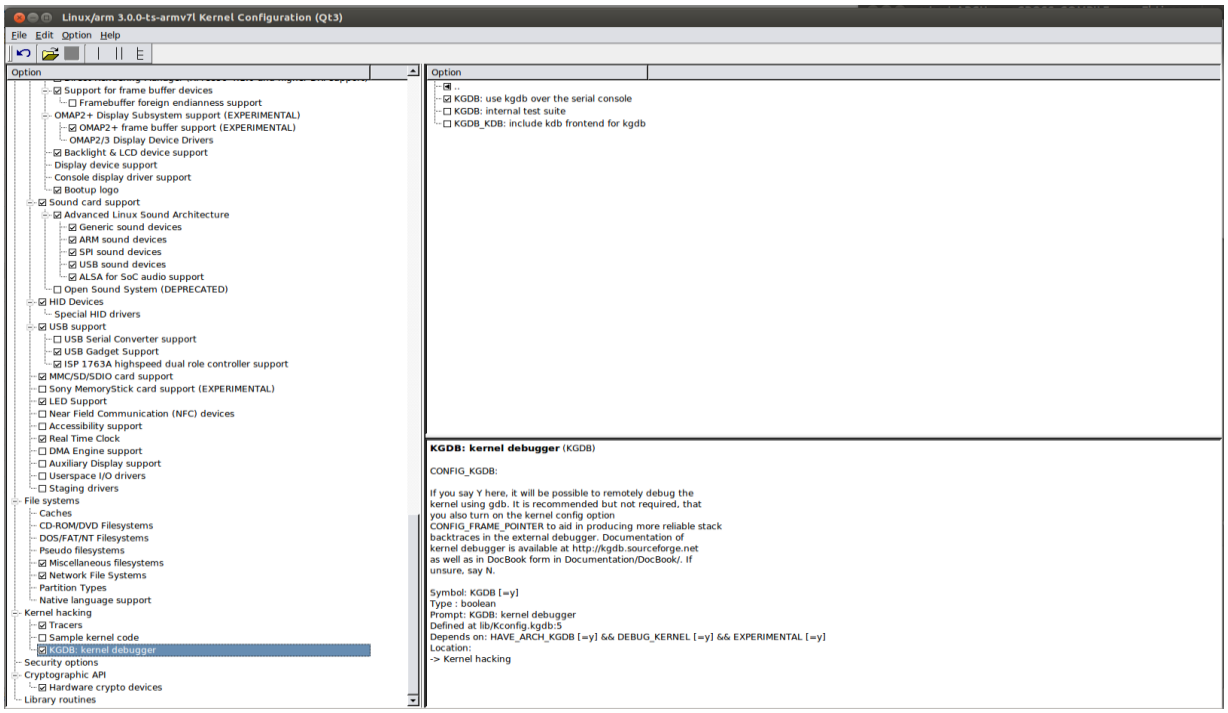
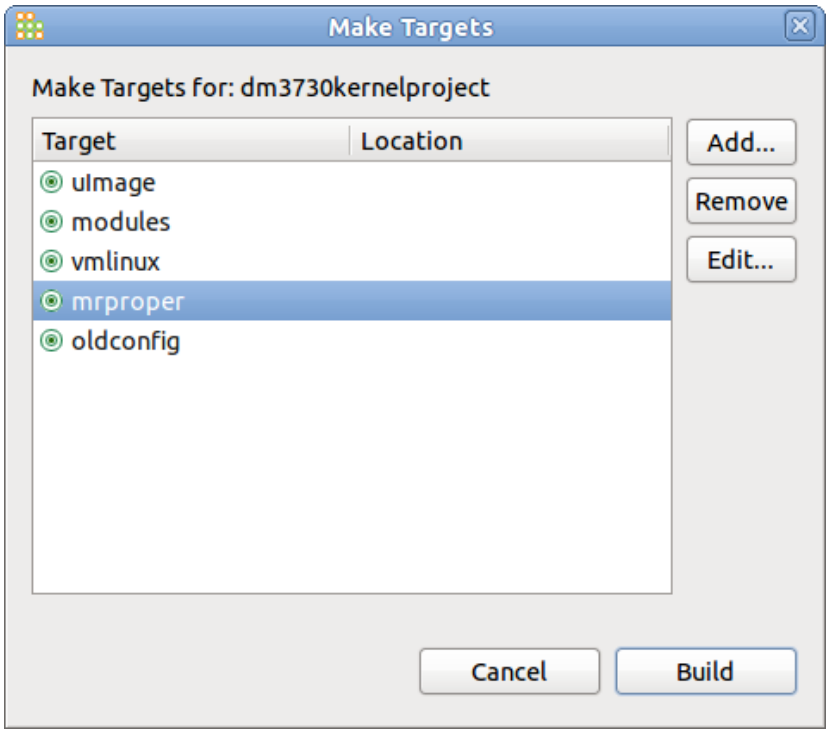


Figure 36: XConfig – Kernel configuration

Building and Deploying the Kernel

To build a kernel project right click on the project and click on build project. The build progress is displayed in the console view.

While building the kernel, sometimes you may see that you have to run mrproper. You can run mrproper by using the make targets included with TimeStorm kernel project. To launch the Make Targets window, right click on the kernel project and click on Make Targets > Build …, and the make targets window will display as shown below.



To run a make target, select the target from the list and click Build.

Booting the board with the kernel image that is built is specific to the board. Please refer the getting started guide for your board for instructions on how to deploy the kernel image and boot the board. TimeStorm does include any features for deploying the kernel and booting the board.

Debugging the Kernel

To debug a kernel, you have to:

- configure the kernel for debugging
- build the kernel
- boot the board with the kernel using the right boot arguments
- debug the kernel from TimeStorm using gdb

Configure the kernel for debugging

To configure the kernel for debugging, open menuconfig or xconfig for the kernel project and:

- In "General Setup", turn on "Prompt for development and/or incomplete code/drivers" (CONFIG_EXPERIMENTAL)
- In "Kernel Hacking", turn on "Kernel Debugging" , "Compile the kernel with debug info", and "KGDB: kernel debugger"

- In "Kernel Hacking" > "KGDB: kernel debugger", turn on "KGDB: use kgdb over the serial console".
- For additional info, refer to https://www.kernel.org/pub/linux/kernel/people/jwessel/kdb/CompilingAKernel.html#CompileKGDB

Build the kernel

After configuring the kernel for debugging, save the kernel configuration and build the kernel.

Boot the Board with the kernel using the right boot arguments

The additional boot arguments passed to the kernel include the gdb and kgdb communication setting, and instructing the kernel to wait for the gdb connection. Gdb communicates to kgdb over serial connection and you set the serial device and the baud rate as the boot arguments.

For kernel debugging append these arguments to your boot arguments.

```
kgdboc=<serial_device>,<baud> kgdbwait
```

For additional information on the boot arguments, refer to https://www.kernel.org/pub/linux/kernel/people/jwessel/kdb/kgdbKernelArgs.html

When you boot the kernel with these boot arguments, kgdb registers the kgdboc driver and waits for a remote gdb connection, as shown below in *figure 37.*

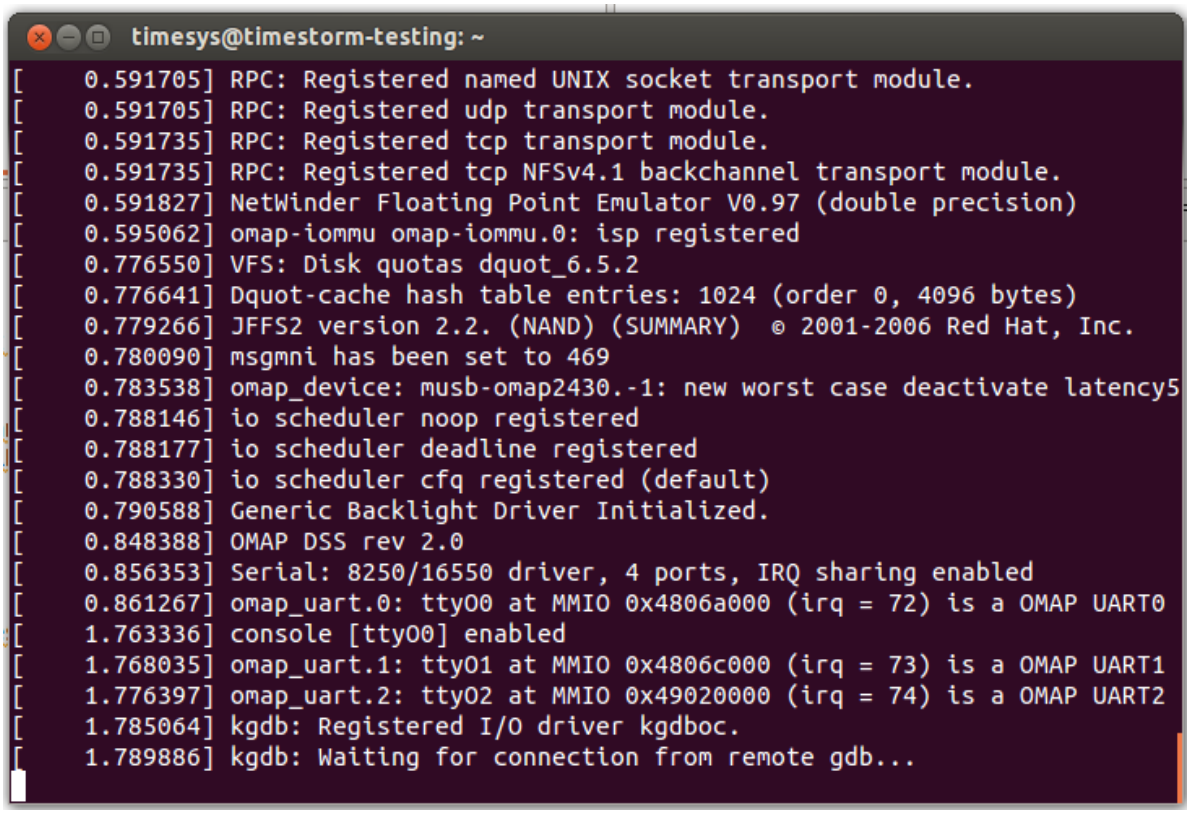


Figure 37: Kernel boot console log – kgdb waiting for gdb connection

Debug the kernel from TimeStorm using gdb

To debug a kernel,

1. Set a breakpoint :
    a. Open the kernel source you want to debug in the source editor
    b. Set a break point by either double clicking at the far left end of the line in the source editor or by right clicking at the far left end of the line in the source editor and click "Add Breakpoint".
2. Create a debug launch configuration and start debugging.
    a. Click and select the kernel project you want to debug
    b. Right click on the kernel project and click on Debug As > Debug Configurations to open the Debug Configurations dialog.
    c. Select Kernel Debugger in the left panel.

d. Click New launch configuration icon in the top left corner. This will create a new debug launch configuration for the kernel project and the launch configuration tabs are displayed in the right panel as shown in *figure 38*. For a kernel that is already built, most of the entries in all the fields are already filled in and you may not have to change any entries for debugging. You have to set the gdb connection settings in the "Connection tab" in the "Debugger" tab.

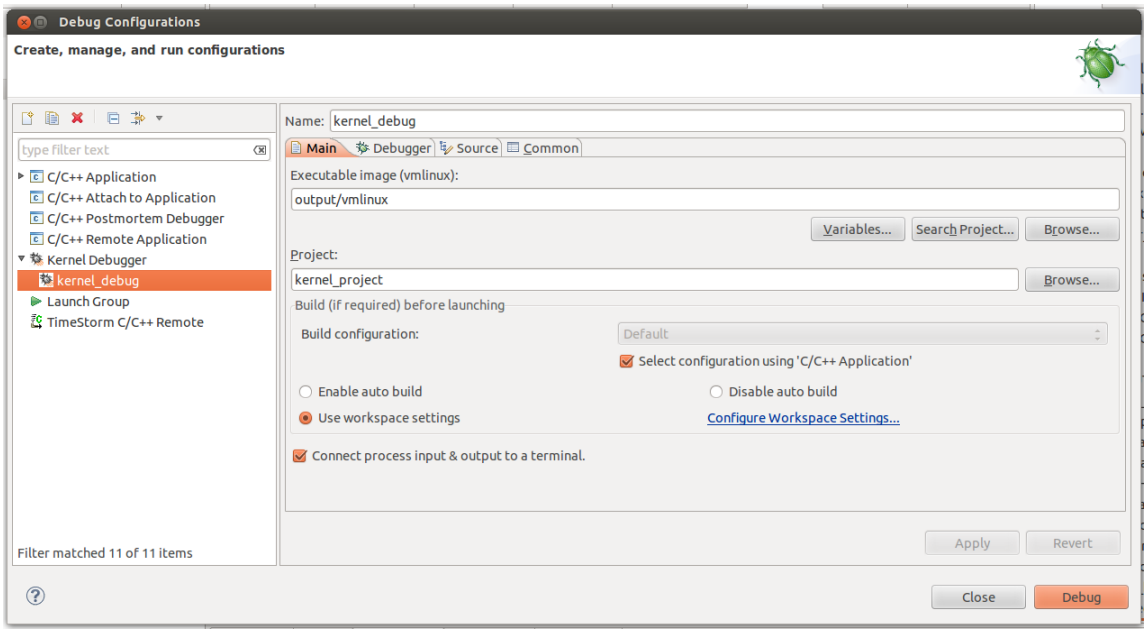


Figure 38: Debug launch configuration for kernel project

Main tab: Enter a name for the debug launch configuration. Enter the kernel project to debug by your workspace by clicking the browse button. Select and enter the vmlinux file in your kernel project.

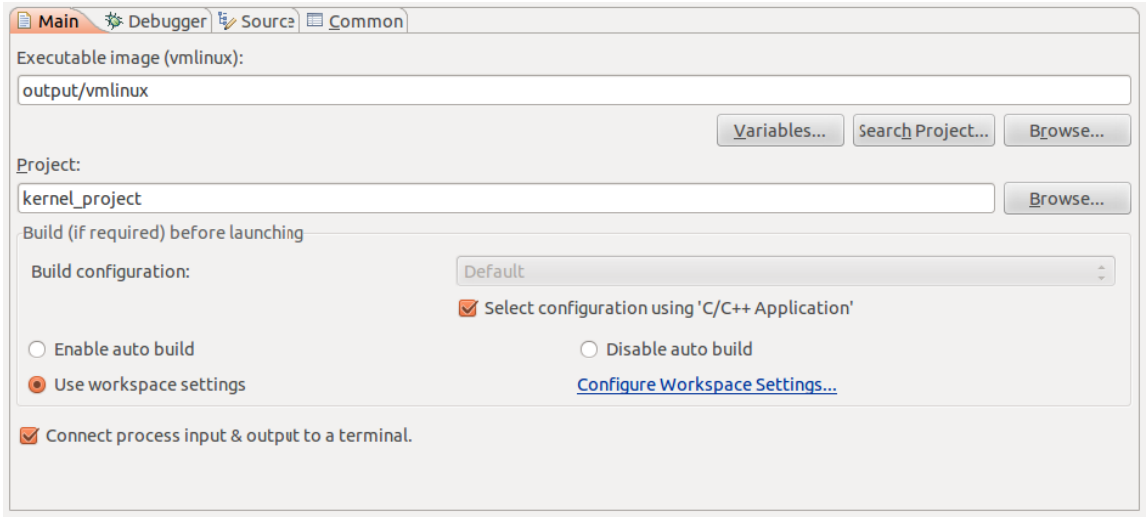


Figure 39: Kernel Debug Configuration – Main tab

Debugger Tab: If you want to change the gdb used to debug the kernel, then you can select the toolchain from the dropdown box at the top of the tab.

Main tab: The gdb in the toolchain is filled in the GDB debugger text field. If you want to change the gdb, you can do so by clicking the Browse button next to the gdb debugger text field. If you wish not to run any commands when gdb launches, you can clear ".init" entry in the gdb command file text field. If you would like to view verbose messages in the gdb console, select the "Verbose console mode" checkbox.

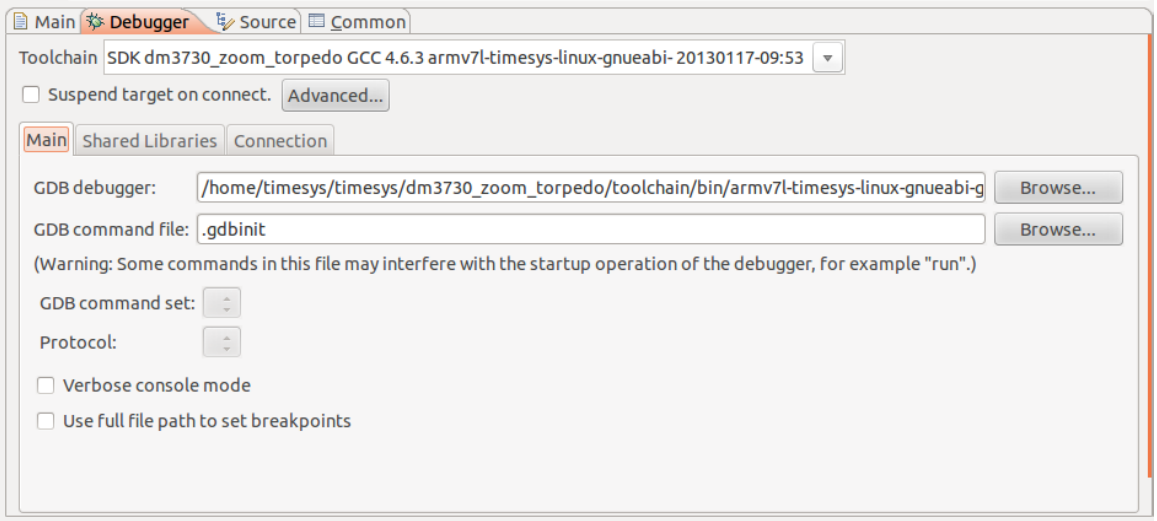


Figure 40: Kernel Debug Configuration: Debugger Tab – Main Tab

Shared Libraries Tab: You can manage the directories the debugger searches to find the shared libraries (with debug symbols) on the "Shared Libraries" tab in the Debugger tab. By default the debugger will automatically search the paths specified in the project's build settings.

Figure 41: Kernel Debug Configuration: Debugger Tab –Shared Libraries Tab

"Load shared library symbols automatically" check box is selected by default and this will allow the debugger to hit breakpoints in the shared library. Select the **"**Stop on shared library events" if you want the debugger to stop on shared library events, even before it hits breakpoints in source code.

Connection Tab: kgdb works only over serial port. So select Serial as the connection type, enter the host serial device that is connected to the target board and the baud rate. This baud rate should match the baud entered in the kernel boot argument.

Figure 42: Kernel Debug Configuration: Debugger Tab – Connection Tab

e. Click the debug button to start debugging. TimeStorm will switch to debug perspective and the debugger will stop at the first break point. You will be able to step in, step over and resume through the kernel source code.

Loadable Kernel Module

A loadable kernel module is a special type of C/C++ project used to write device drivers. The loadable kernel module is developed referring specific kernel sources and kernel image and the board should be booted with the same kernel image.

Creating a Lodable Kernel Module Project

Top open a new loadable kernel module project click File > New > Project > C/C++, click Next and choose Kernel Module as the project type as shown in *figure 43.*

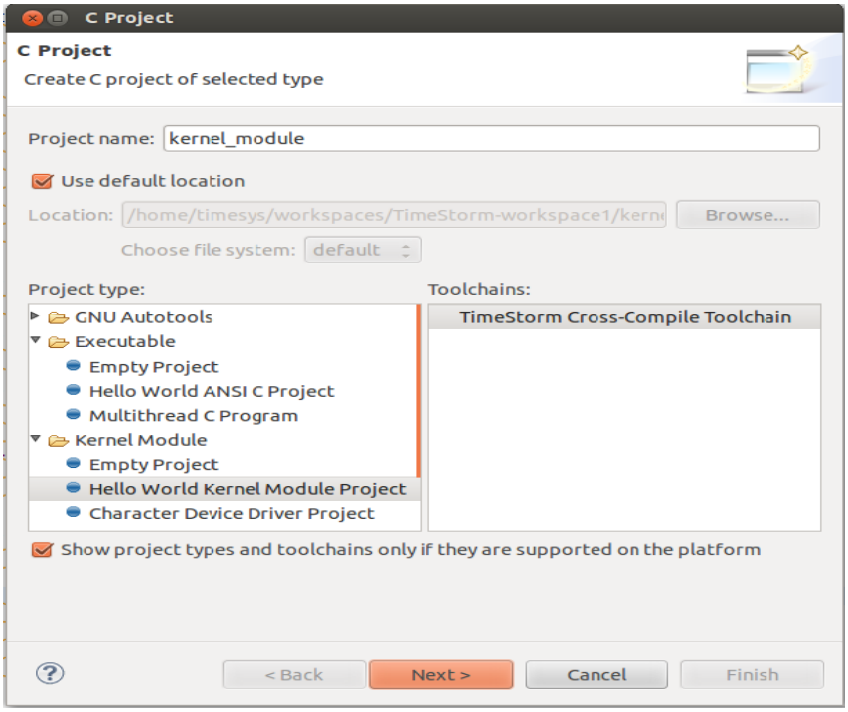


Figure 43: Loadable kernel module project

You can start developing a kernel module project from two of the templates included with TimeStorm, or you can start from an empty project. Enter a name for the project and click next.

Select the build configurations to be created for the project as shown in figure 44 and click Next.

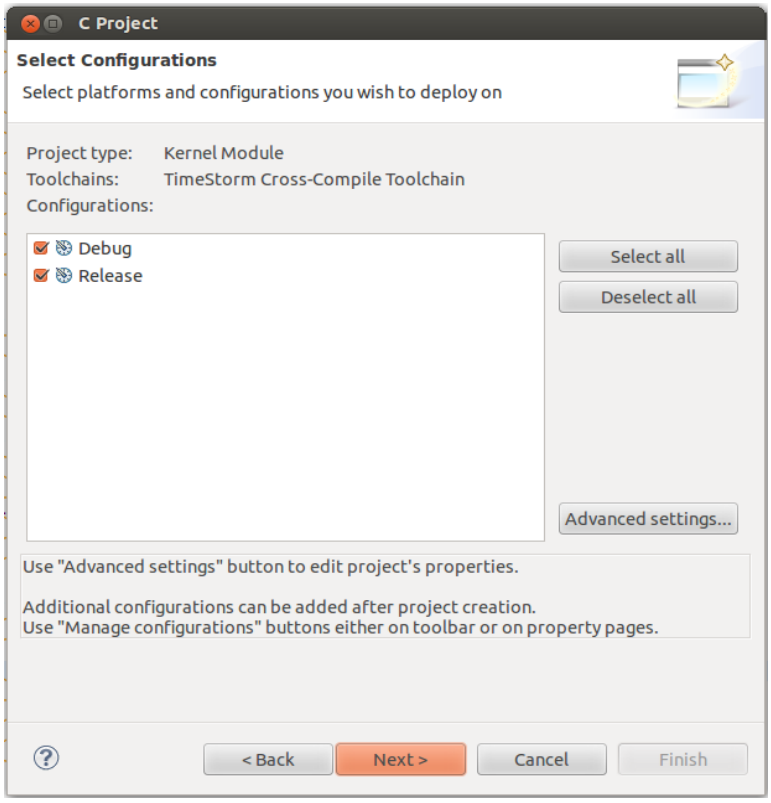


Figure 44: Loadable kernel module project – Build configurations

Select the toolchain you want to use with the kernel module as shown in *figure 45* and click next. Be sure to select the toolchain that is used with the kernel project / kernel sources you will be choosing on the subsequent page.

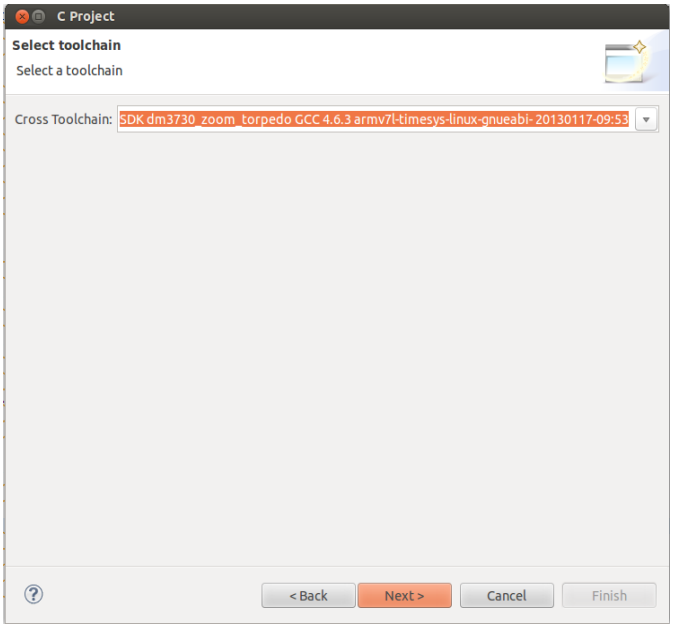


Figure 45: Loadable kernel module project – Toolchain selection

Kernel module development requires the location of the sources and a kernel image. You can choose a kernel project that is already created in your workspace (recommended) or you can choose an external directory that contains the kernel sources as shown in *figure 46.*

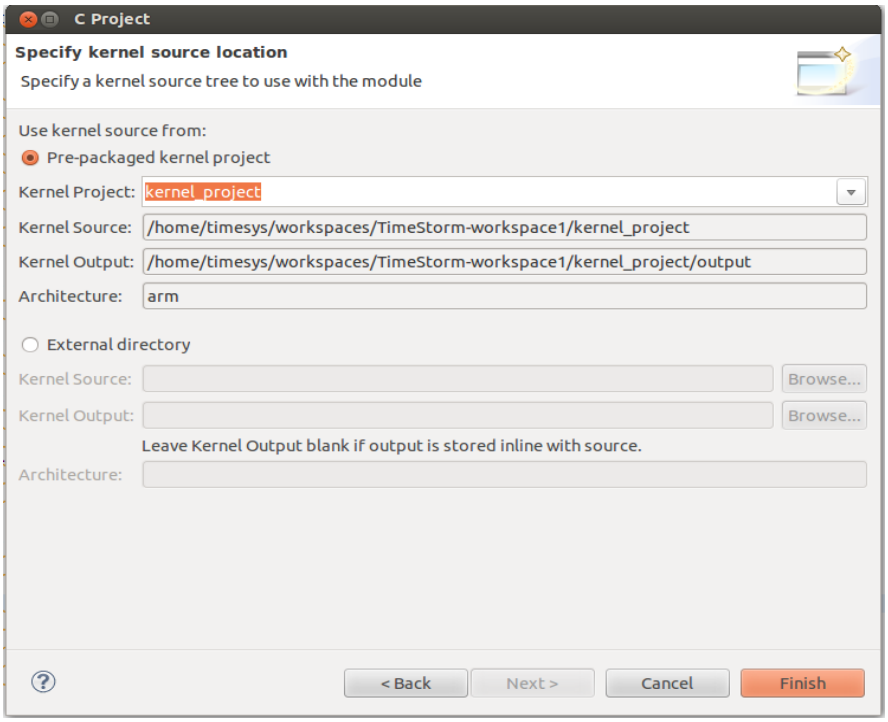


Figure 46: Loadable kernel module project – Kernel source location

Click finish to create the kernel module project.

Building Kernel Module Project

A loadable kernel module project is built the same way as other C projects. To build the kernel module, choose a build option from Project menu or from the context menu that appears when you right click on the project name.

Deploying the Kernel Module

To deploy a kernel module on a running target, you will create and run a kernel module launch configuration. To create a kernel module launch configuration, select the kernel module project and click Run > Run configurations or right click on the kernel module project and click Run As > Run Configurations.

Click Kernel Module in the left panel and click New icon in the top left corner.

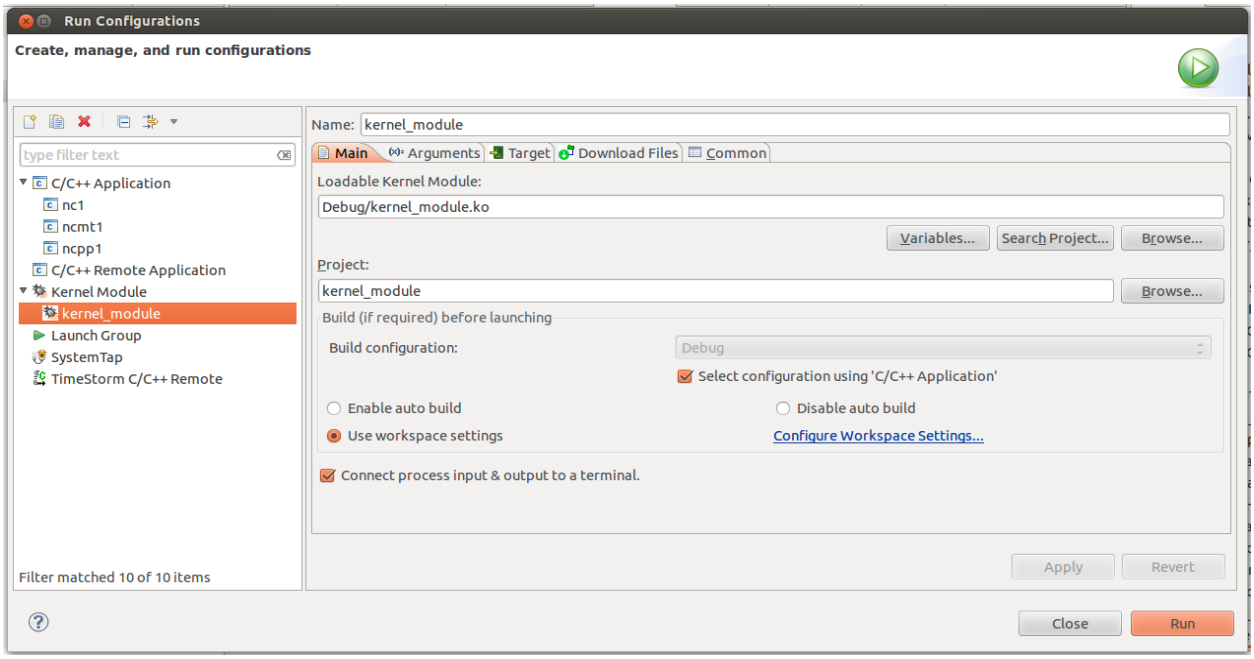


Figure 47: Kernel Module launch configuration

Enter a name for the launch configuration. The module to be deployed to the target and the project name are filled in. You can change them if you want to make any modifications.

The kernel module is deployed to the target by using a mod_install.sh script. This script is filled in as the command to execute on the arguments tab. The kernel module is the argument to this script, which is also filled in the program arguments text field.

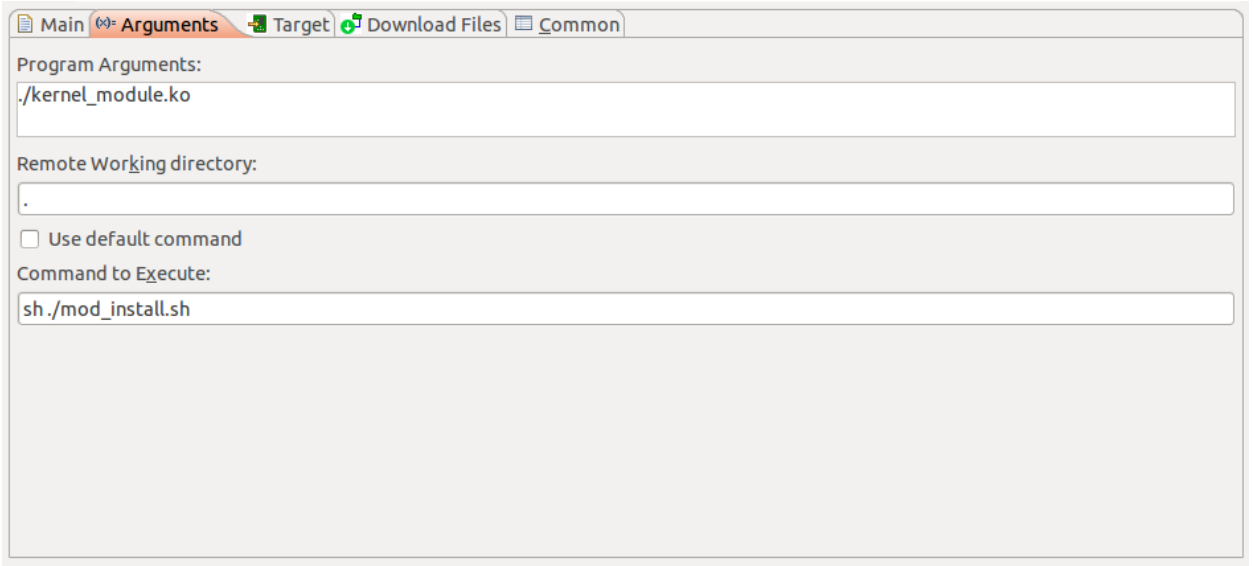


Figure 48: Kernel module launch configuration – Arguments tab

Other tabs in the launch configuration are similar to those explained in the C/C++ application debugging.

Qt Development with TimeStorm

Qt is a cross-platform application and UI framework with APIs for C++ programming. TimeStorm is bundled with the Qt Eclipse integration plugins. These plugins will help in developing Qt applications using a Timesys SDK, and remotely running and debugging the Qt applications on target. Timesys SDKs that include Qt are configured to work easily with TimeStorm. These are a few main steps that get you started developing Qt applications using TimeStorm:

1. **Switch to Qt perspective:** Click *Window > Open Perspective > Other > Qt C++*

2. **Configure TimeStorm for Qt:** To create Qt projects and develop Qt applications, you have to configure TimeStorm for Qt by adding the Qt version you want to use. You can add multiple Qt versions to TimeStorm – native Qt, Qt specific to your board. To add a Qt version, click *Window > Preferences > Qt > Add*. A dialog *(as shown below in Figure 49)* is opened.

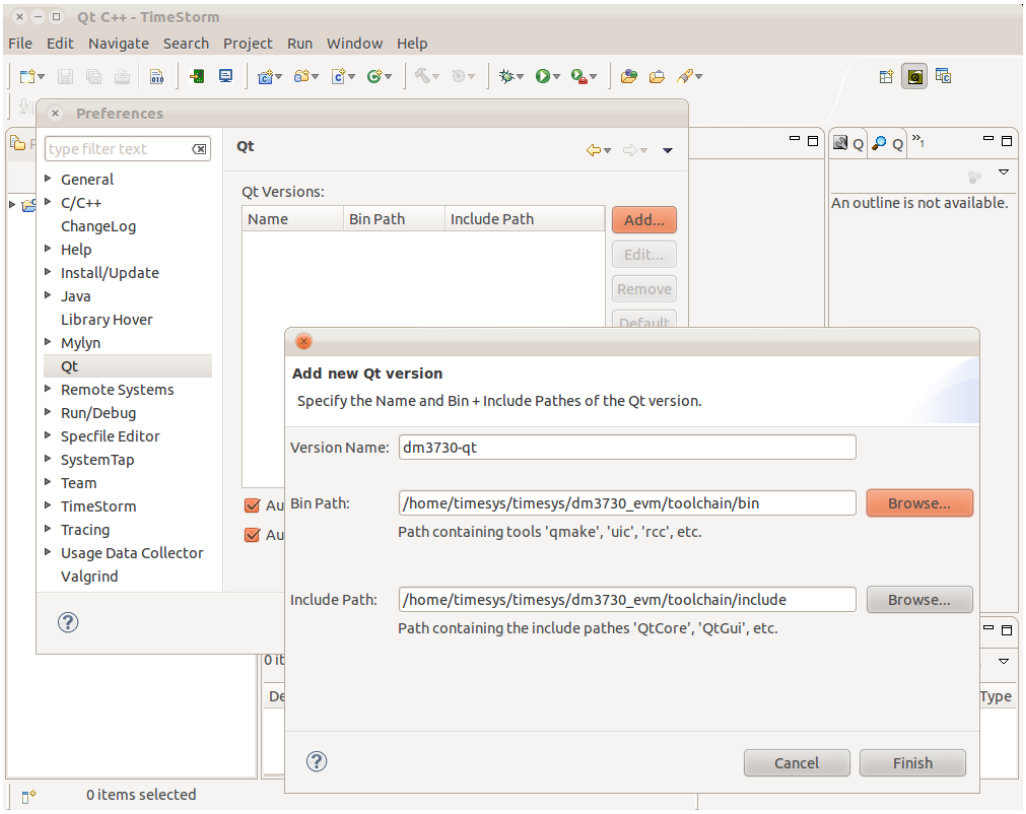


Figure 49: Configuring Qt

3. **Enter a name for the Qt version**

4. **Select the Bin Path** — Click 'Browse' to select the bin folder of the toolchain that includes Qt.

5. **Include Path** — The include path will be automatically set: you need not change it.

6. **Click the 'Finish' Button.** Now TimeStorm is set to use the Qt version you have added.

To Create a Qt project and develop a Qt application:

1. Click *File > New > Project > Qt > Qt Console Project / Qt Gui Project* to launch the project wizard
2. Fill in the project name and other details
3. Click the 'Finish' button.

During development, you may want to first develop and debug your Qt apps on the host and then rebuild them for the target board.

To switch the Qt version:

1. Right-click on the project, and click *Properties > Qt Properties.*
2. Select the Qt version you want to use *(as shown in Figure 50).*

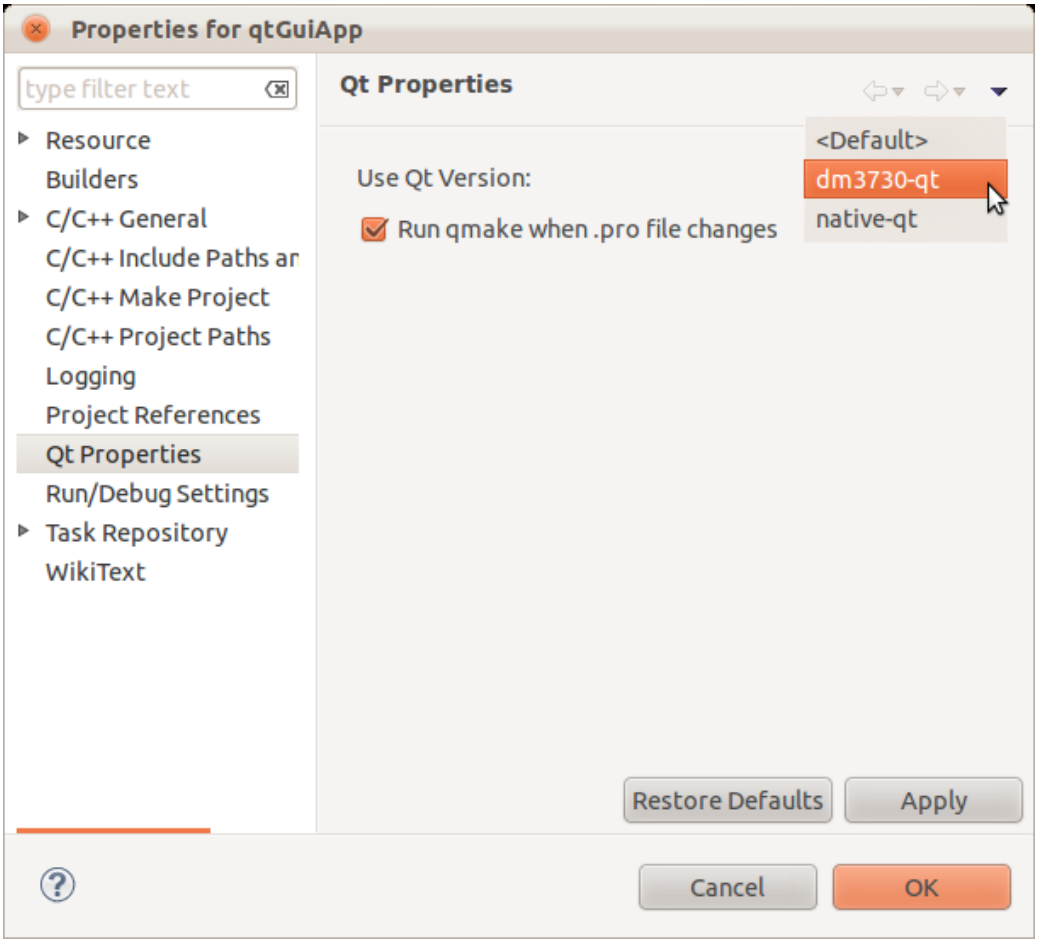


Figure 50: Changing Qt version for a project

After booting the board with the SDK that includes Qt, you will be able to remotely run / debug the Qt apps on the Hardware target.

To Run / Debug Qt an application on the Target:

1. Follow the procedure explained in Run / Debug Configurations.

QEMU Launcher

QEMU Launcher will allow you to select a QEMU enabled SDK and run QEMU. QEMU launcher dialog can be opened by clicking the icon in the toolbar or by clicking Run > Launch QEMU. The launcher dialog will be displayed as show in *Figure 51*.

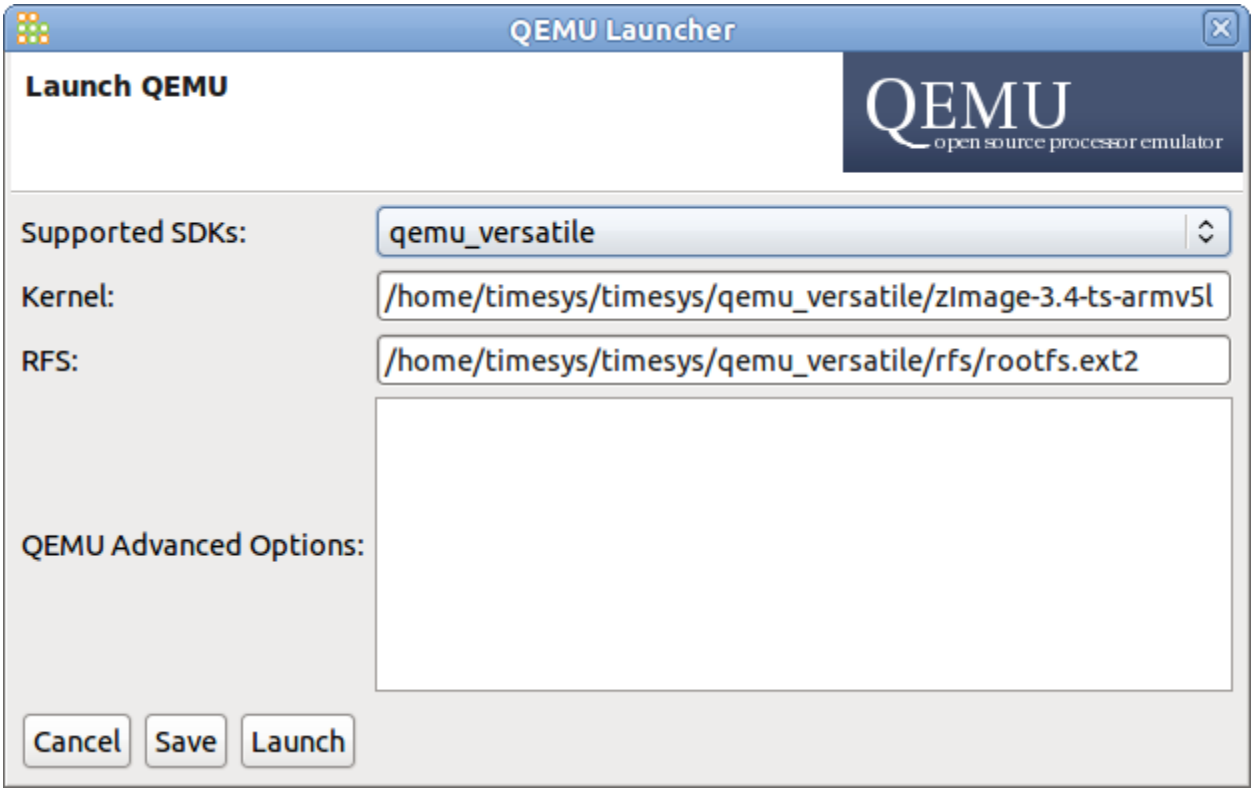


Figure 51: QEMU Launcher Dialog

Launch QEMU: QEMU launcher will detect the QEMU enabled SDKs and populate them in the Supported SDKs drop down box. To launch QEMU, select an SDK. The kernel and RFS for QEMU are pre filled from the selected SDK. If you want to launch QEMU with a different kernel or RFS, you may edit the kernel and RFS path. If you want to pass additional options to QEMU, you may enter it in the QEMU Advanced Options text area. Click the launch button to launch QEMU. QEMU will be launched in a terminal window and you will be required to enter your sudo password as shown in *Figure 52*.

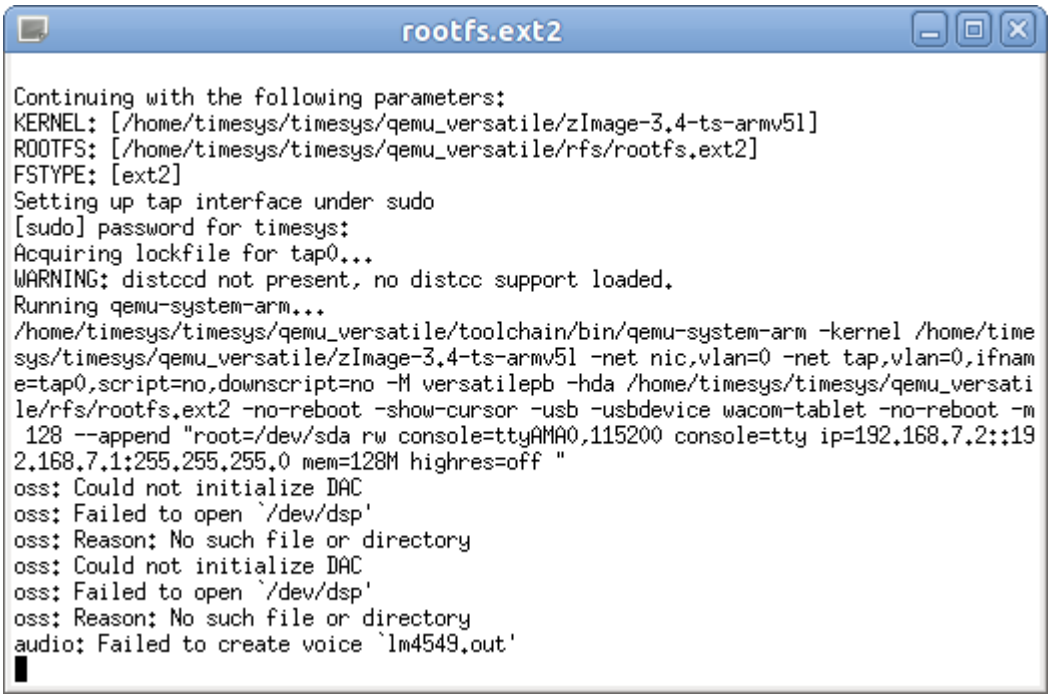


Figure 52: QEMU terminal window

After entering the sudo password, QEMU will run in a separate graphical window as shown in *Figure 53*. If you wish not to run QEMU in a separate graphical window, but continue to run in the terminal window as in *Figure 52*, enter `-nographic` in the advance parameters text field in the launcher dialog.

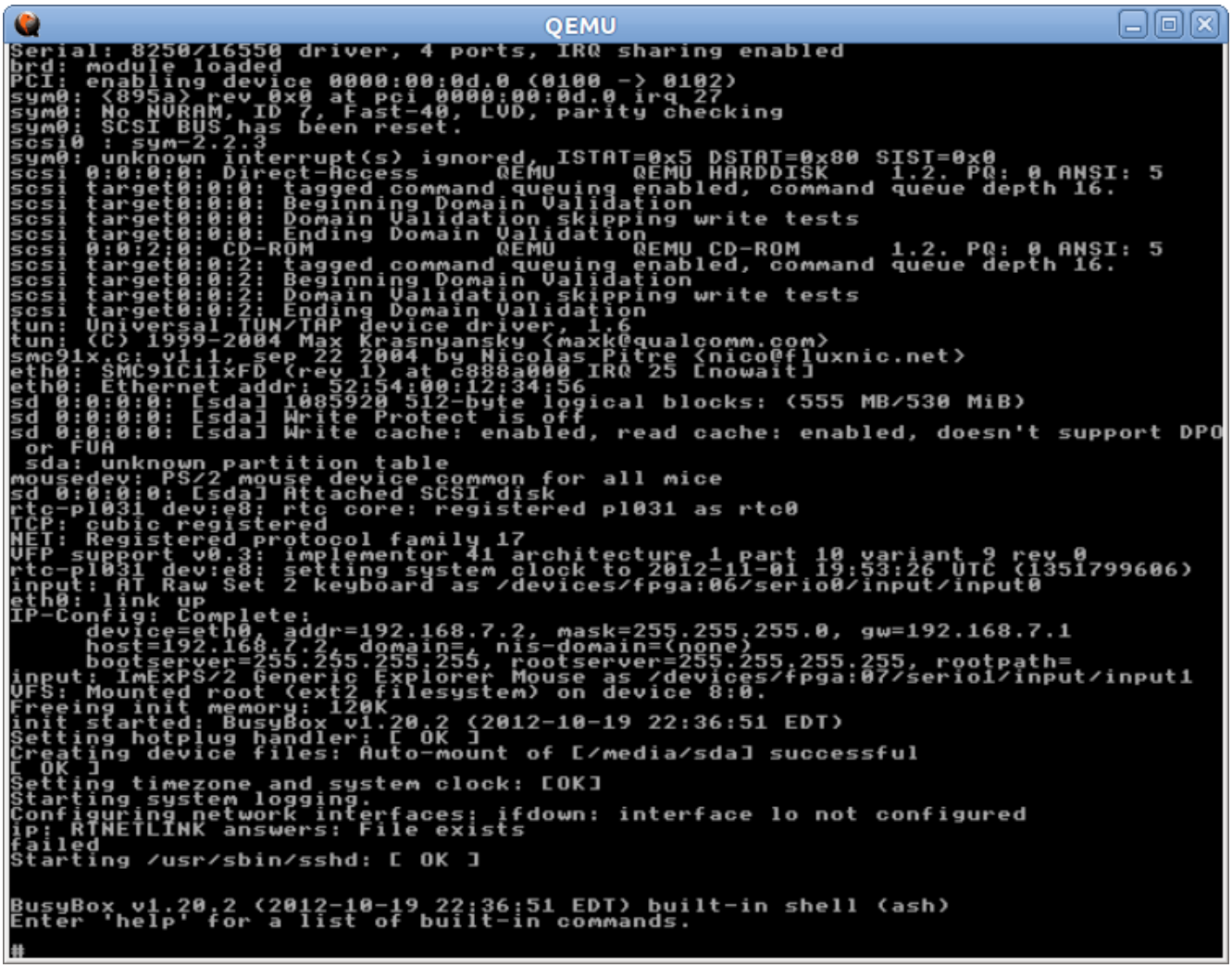


Figure 53: QEMU graphical window

Run / Debug application in QEMU: To run / debug an application in QEMU you have to create a hardware target definition for QEMU. By default, QEMU is assigned an IP address of 192.168.7.2. Set the root password for QEMU, and use them to define a hardware target as explained in Working with Hardware Targets. Then you can use this QEMU to run / debug your application.

Stopping QEMU: To stop QEMU type `reboot` in the QEMU window. You may have to enter the sudo password again in the QEMU console. You can close the QEMU console window after QEMU has terminated.

Remote System Explorer

Remote System Explorer (or RSE in short) is the new addition from Eclipse for remote target interaction. RSE allows you to define targets, explore target file system, and run and debug applications on target. Timesys recommends using the 'Hardware Target' functionality provided by Timesys as explained in the Working with Hardware Targets section, because:

- RSE using SSH does not work with dropbear
- RSE does not support serial connection
- RSE does not support NFS

To use RSE with boards running Timesys SDK:

- Include openssh package in your target RFS. RSE requires a sftp server on the target and the dropbear ssh server does not include an sftp server.
- When defining a connection in RSE, choose 'SSH only' system type.

After you define a target connection in RSE, you will be able to browse the target file system, copy files between local host and target, and open a terminal console to the target.

For additional information on RSE, refer to the Help included in TimeStorm by selecting *Help > Help Contents*.

Other tools

TimeStorm bundles other useful tools from the Eclipse community http://www.eclipse.org/linuxtools For detailed help on how to use the tools, please refer to the Help included in TimeStorm by selecting *Help > Help Contents*. Below is a summary of the tools:

- **Gprof —** Profile an application using gprof and visualize the gmon.out files generated by gprof instrumentation. To learn how to use gprof with TimeStorm, read https://linuxlink.timesys.com/docs/wiki/engineering/HOWTO_Use_Gprof_with_TimeStorm.

- **Gcov —** Test code coverage in program using gcov and visualize the gcda and gcno files generated by gcov intrumentation. To learn how to use gcov with TimeStorm, read https://linuxlink.timesys.com/docs/wiki/engineering/HOWTO_Use_Gcov_with_TimeStorm.

- **LTTng —** Linux Trace Toolkit, is a high-performance tracing tool for Linux that efficiently handles large amounts of trace data. Initially focused on the Linux kernel, its technology has been extended to support user space tracing (UST). From TimeStorm, you can configure and control LTTng, collect the trace data, and visualize and analyze the trace data.

- **OProfile —** A powerful profiling tool and using TimeStorm you will be able to configure OProfile, gather OProfile data and visualize it. In addition to the OProfile tool from Eclipse, TimeStorm includes a "Linux Profiling Suite" for easily profiling applications using OProfile. Linux Profiling Suite is explained in the next section.

- **Valgrind —** A powerful tool that can be used from TimeStorm for profiling applications. TimeStorm adds support to use valgrind on a remote target.

Linux Profiling Suite (LPS)

TimeStorm supplements the Eclipse OProfile tool with its own Linux Profiling Suite (LPS). LPS is easy to use and well integrated with TimeStorm hardware targets management.

LPS allows embedded developers to find performance bottlenecks in their code. LPS uses OProfile, which is Open Source profiler for Linux systems. OProfile collects information about kernel and all running applications. OProfile profiles your code by recording the output from hardware registers that counts events such as CPU cycles. LPS provides statistical analysis of profile results and annotated source code view, helps in identifying any performance issues easily and quickly.

For more information about OProfile, refer to the OProfile Manual found at http://oprofile.sourceforge.net/doc/index.html

Note: You must have OProfile installed on target for remote profiling or on host machine for local profiling.

LPS Perspective

Select *Window > Open Perspective > Other* option from main menu and choose 'Linux Profiling Suite' from 'Open Perspective' window (*shown in Figure 54*).

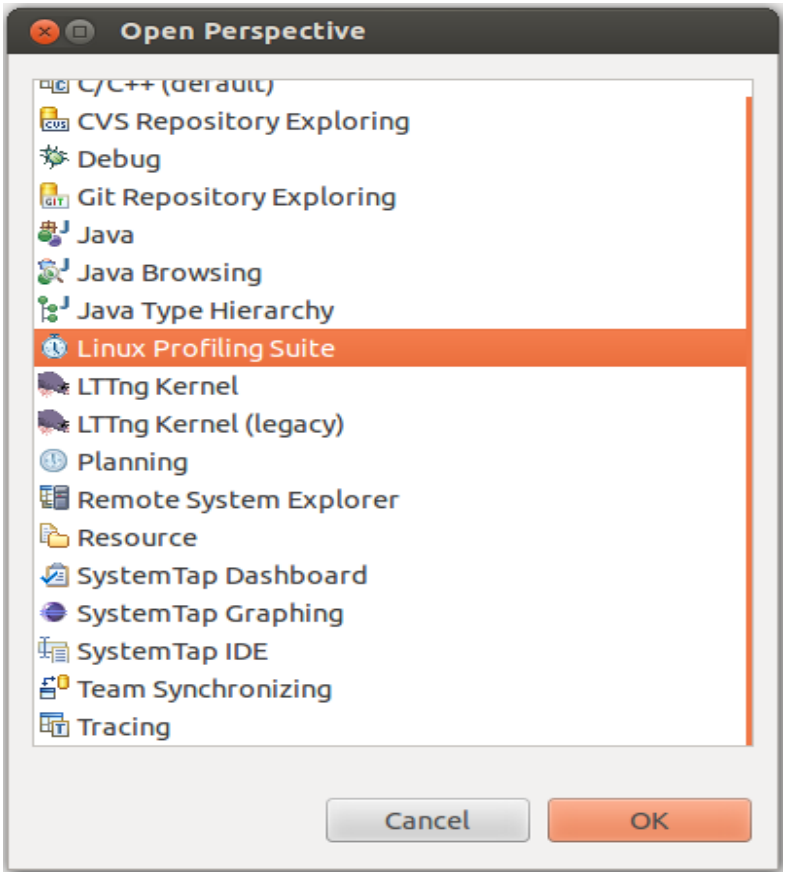


Figure 54: Open Perspective Window

The LPS perspective shown in *figure 55*, includes:

- Profiling Monitor view – to organize and manage profiling sessions
- Profile Analysis view – to view and analysze the OProfile profile data
- Properties view – to view the properties of the actively selected node in the Profiling Monitor View
- Profile Settings view – to view the various OProfile settings used for the profiling session

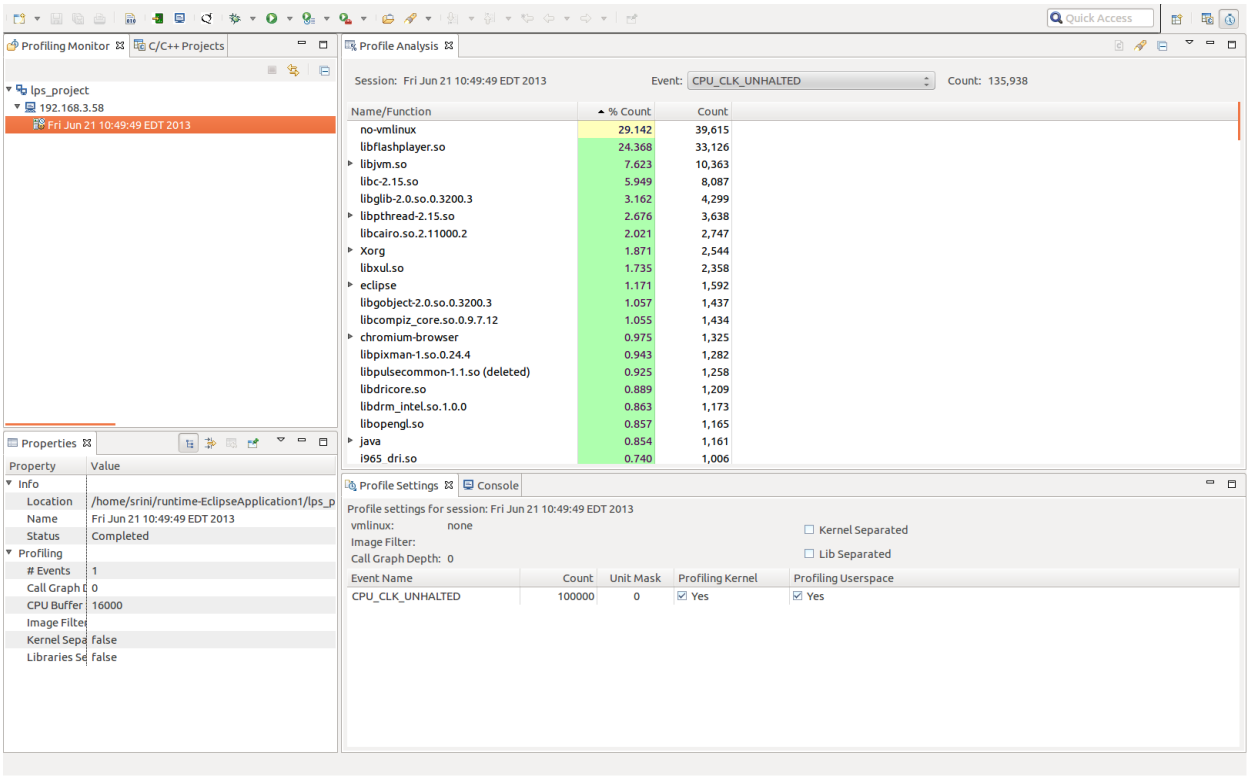


Figure 55: LPS Perspective

Profiling using LPS

To profile an application or system, create a profile launch configuration.

1. In the 'Profiling Monitor' right click and select 'Profiling Tools Configuration' to create 'Profiling Tools Configuration' as shown in *figure 56.*

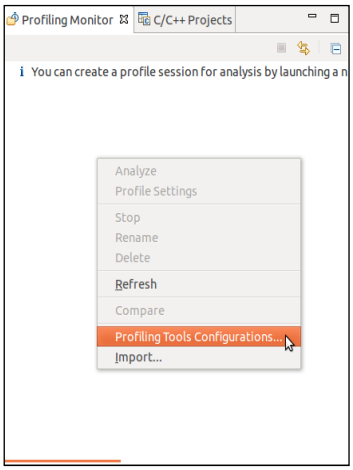


Figure 56: Launching Profiling Tools Configuration

2. Select 'TimeStorm Oprofile' in left panel.
3. Click New launch configuration icon in top left corner. This will create a new profile launch configuration and the launch configuration tabs are displayed in the right panel (shown in *figure 57)*

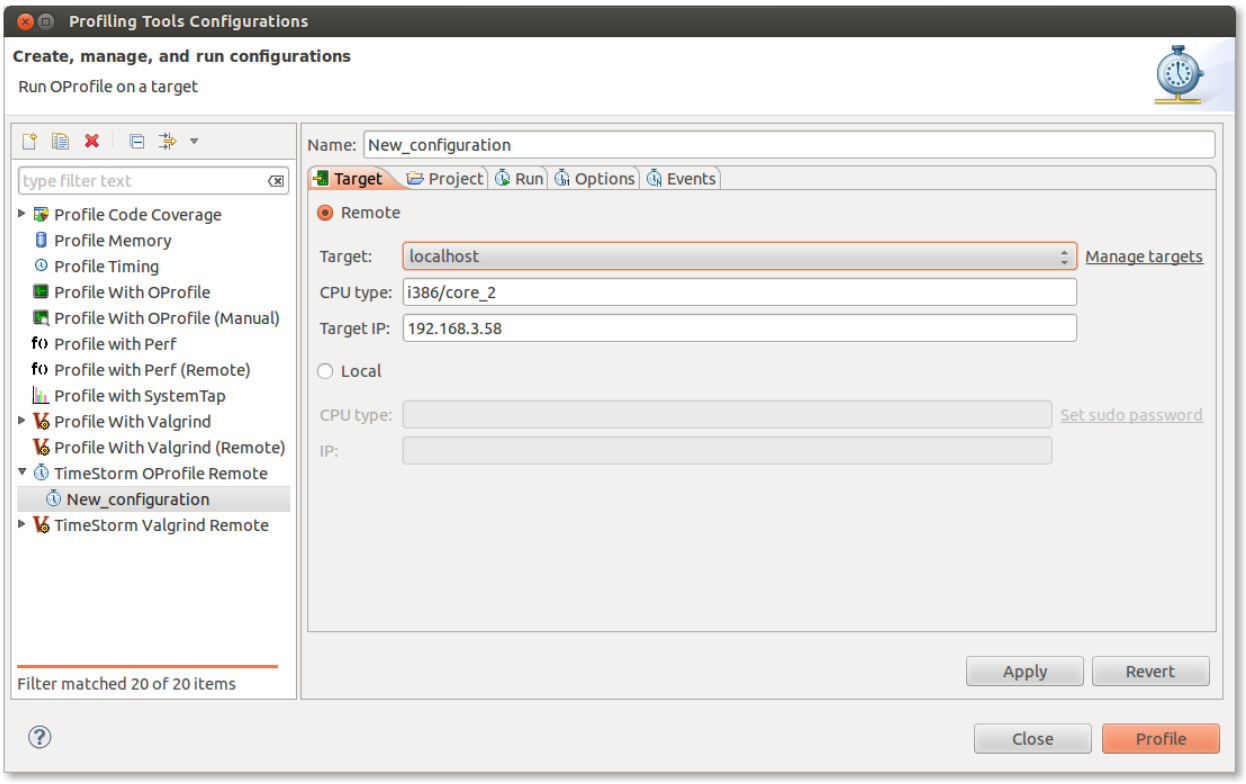


Figure 57: Creating a profile launch configuration

Target Tab

The target tab is show in *figure 58.* Choose "Remote" to run OProfile on remote target or choose "Local" to run OProfile on local host. When the target is selected, TimeStorm checks the OProfile CPU type and fills in the details. An error message is displayed if TimeStorm cannot run OProfile commands.

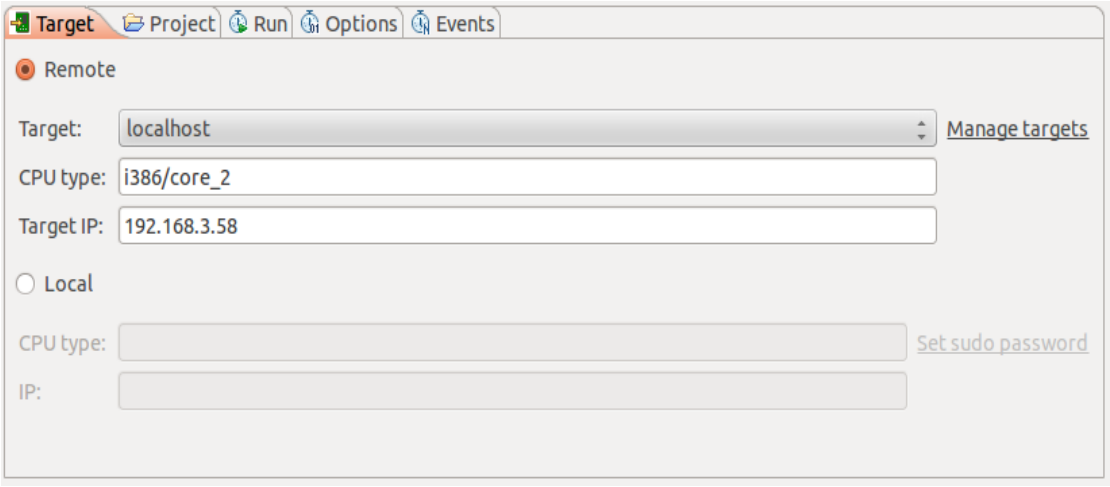


Figure 58: LPS Configuration – Target Tab

Project Tab

The project tab is shown in *figure 59.* LPS organizes profiling sessions into projects and sessions. You can create a new profiling project or choose an existing one. The session name is optional. If session name is left blank, TimeStorm creates a session using the date and time.

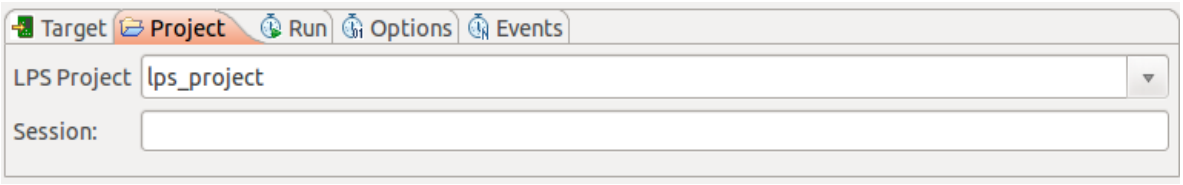


Figure 59: LPS Configuration – Project Tab

Run Tab

The run tab is shown in *figure 60.* LPS allows you to profile an application or the whole system for a specified duration.

Application – If you want to profile an application, select it from either your workspace or file system by clicking the Browse button. If you want to skip profiling during application startup, you can delay the OProfile startup by setting the delay time.

Timed – Select this option to profile the whole system for specific duration.

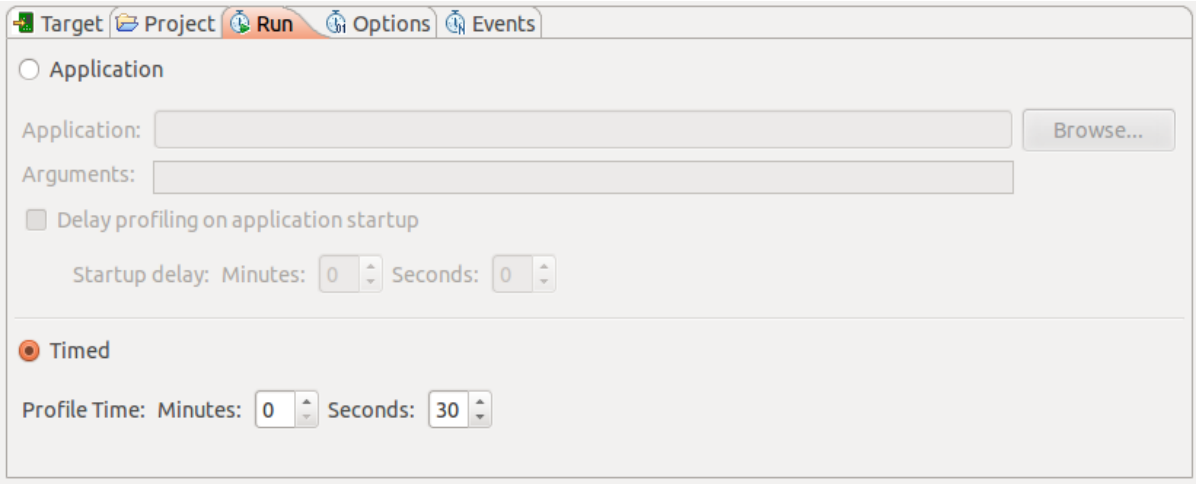


Figure 60: LPS Configuration – Run Tab

Options Tab

This tab allows you to configure OProfile.

Kernel image (vmlinux) – If you want to profile the kernel, select the vmlinux kernel image file by clicking browse. If you do not have a vmlinux file or you do not want to profile the kernel, select 'no-vmlinux' checkbox.

For a detailed explanation of other options on this tab, please refer to:

http://oprofile.sourceforge.net/doc/controlling-daemon.html

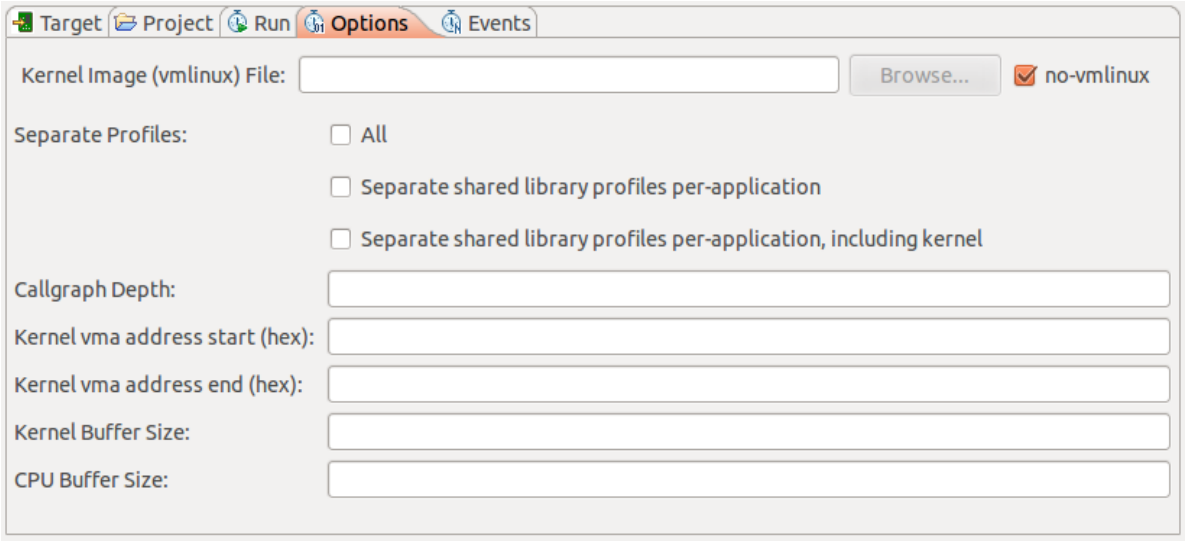


Figure 61: LPS Configuration – Options Tab

Events Tab

This tab allows you to specify the events for each of the hardware performance counters. The OProfile CPU type and the number of performance counters on your target are displayed at the top.

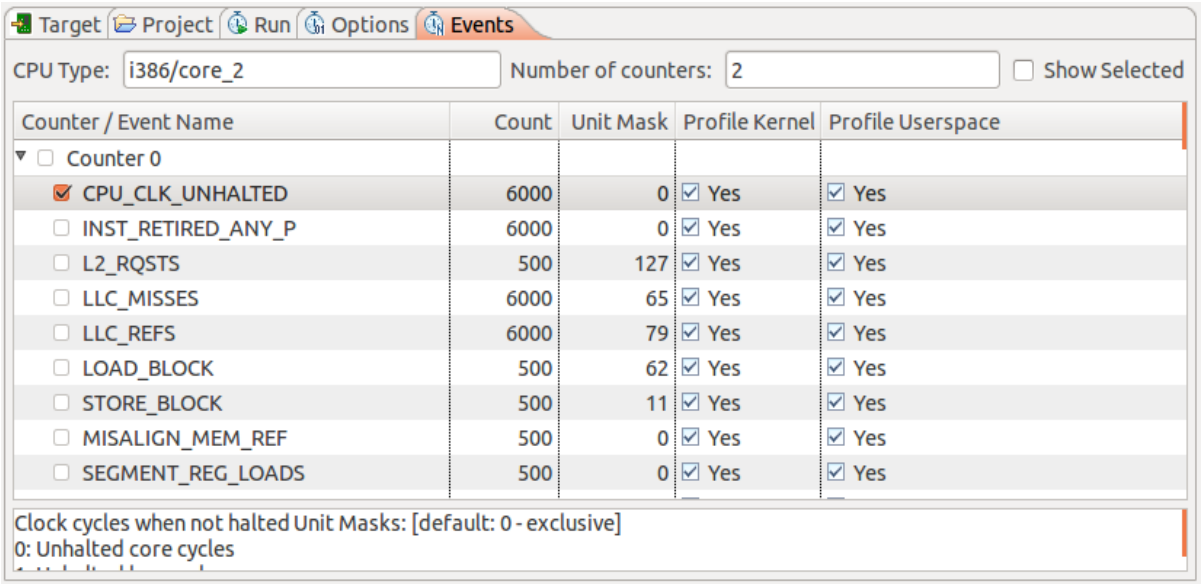


Figure 62: LPS Configuration – Events Tab

Counter / Event Name – Expand the counter to view events in the counter. Select the event for profiling. You may only choose one event per counter. Also, if you choose a particular event for one counter, you may not choose the same event for another counter.

Count – The field shows the counter reset value for the given event. To edit this value, click on counter value, type in new value and hit Enter key.

Unit Mask – This field shows the unit mask. To edit this value, select the event and enter new mask value in 'Unit Mask Editor' (*shown in Figure 63)*

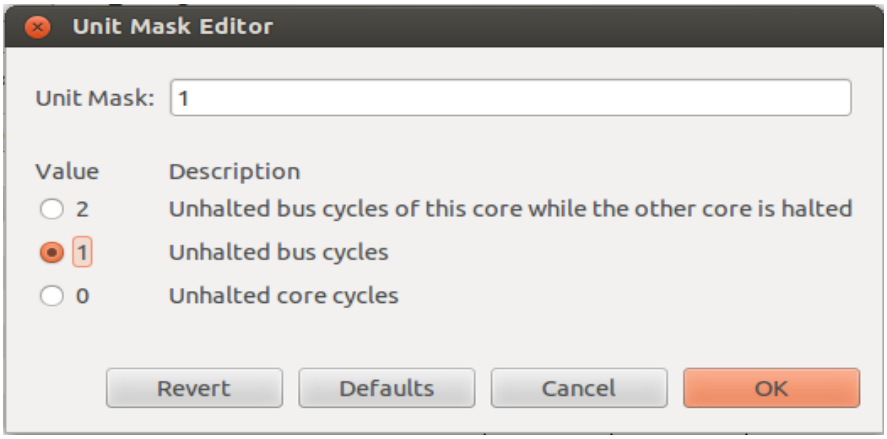


Figure 63: Unit Mask Editor

Profile Kernel – Select this to profile kernel code.

Profile Userspace – Select this to profile userspace code.

For more information on the counters and events, please refer to OProfile documentation.

After configuring OProfile, 'Click' the Profile button to start profiling. The profiling status and progress is displayed in the Profiling monitor view.

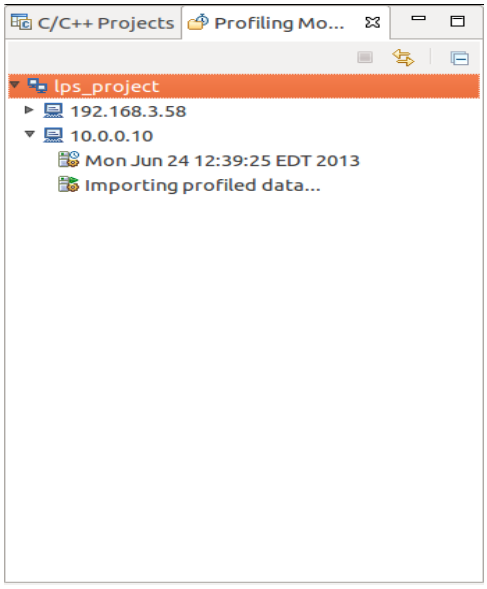


Figure 64: Status messages in Profiling Monitor view

Analyzing the Profiled Data

To view and analyze a successfully completed profiling session, right click on the session and click on "Analyze" as shown in *figure 65.* Alternatively, you can also double click on the session. This will open the profile analysis view and display the profiled data.

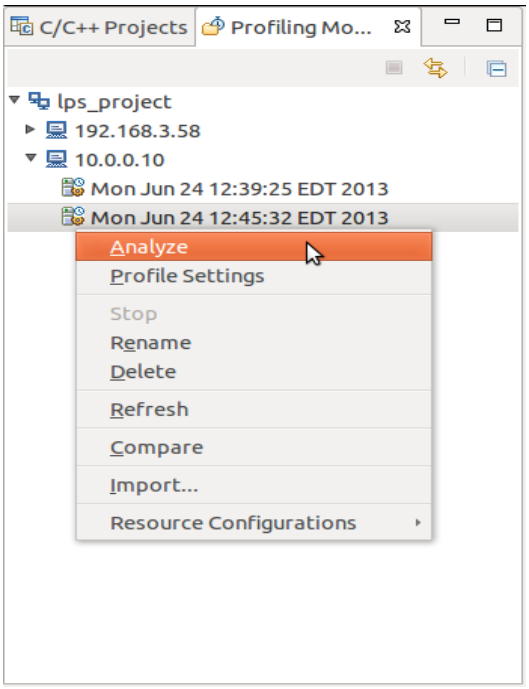


Figure 65: Selecting Analyze option

The profile analysis view displays the session name and the profiled events in a drop down. Selecting a profile event will display the total profile count for that event and the applications, libraries and functions that are profiled for that event. When an executable includes debug information, the filename and the line number are also displayed. For example in the figure, CPU_CLK_HALTED event is selected, multithread application is profiled, the thread_func has the highest profile count at line number 26.

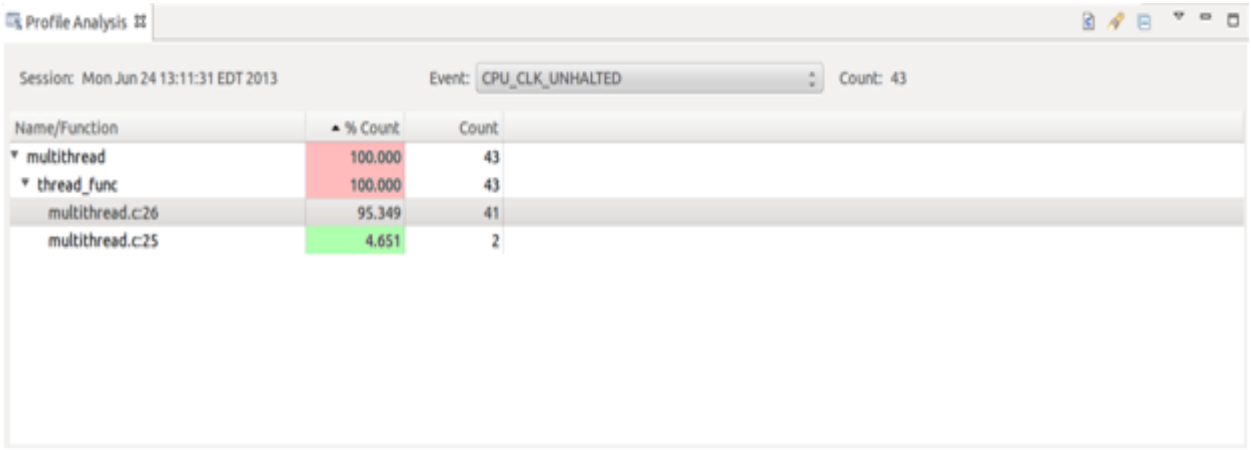


Figure 66: LPS Profile Analysis view

To analyze and drill down through large profile data, you can

- sort by column by clicking on the column header

search in the filename / function by using the search facility by clicking flashlight icon in the analysis view toolbar.

Focus on the functions of interest or critical bottle necks by using the name filter, count filter and percentage count filter. The filters can be configured, set and unset using the down arrow in the analysis view toolbar.

For files that have debug information, you can view the profile data and the source code side by side in the annotated source editor as shown in *figure 67*. You can open the annotated source editor by clicking the 'View Source' button in analysis view's toolbar.

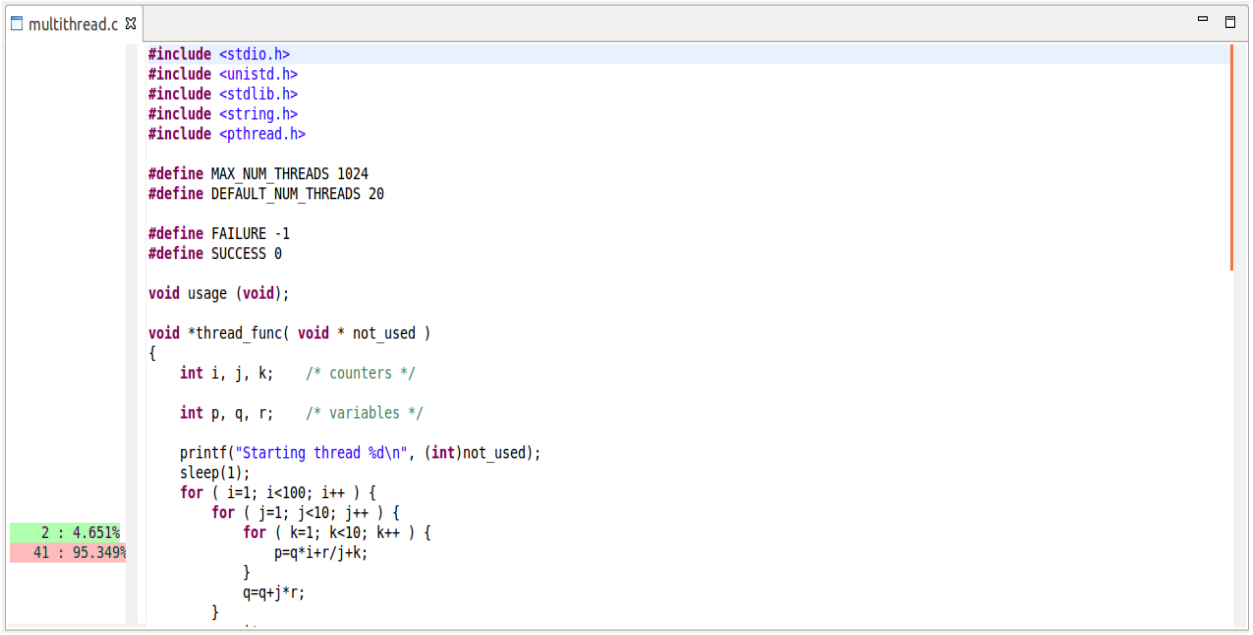


Figure 67: LPS – Annotated C Editor

Profile Settings View

To view the OProfile settings for a profile session, right-click on the session in profiling monitor view and select 'Profile Settings'. The settings are displayed in Profile Settings view as shown in *figure 68.*

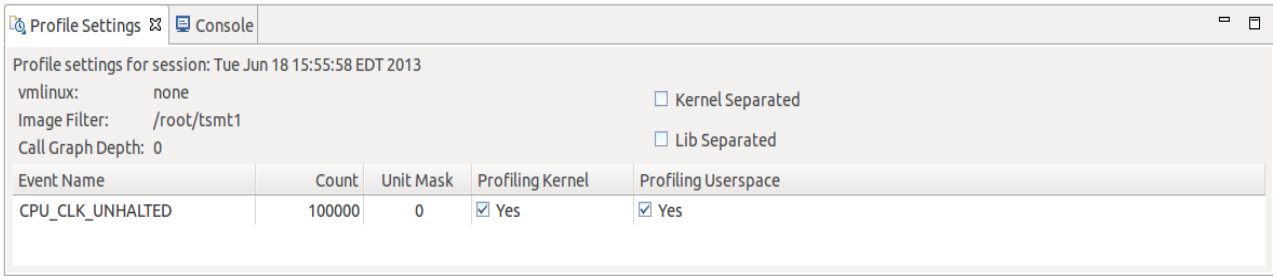


Figure 68: LPS – Profile Settings view

Importing OProfile Data

If you have already profiled your system / application using OProfile and you want to use LPS to analyze the results, you can import the data to create a session for analysis. To import OProfile data,

1. Right click in the 'Profiling Monitor' view and select the 'Import' option from context menu to open Import wizard.
2. Choose *Other > OProfile Data into LPS Project* and then click 'Next' to open the Import dialog (*shown in figure 69*)

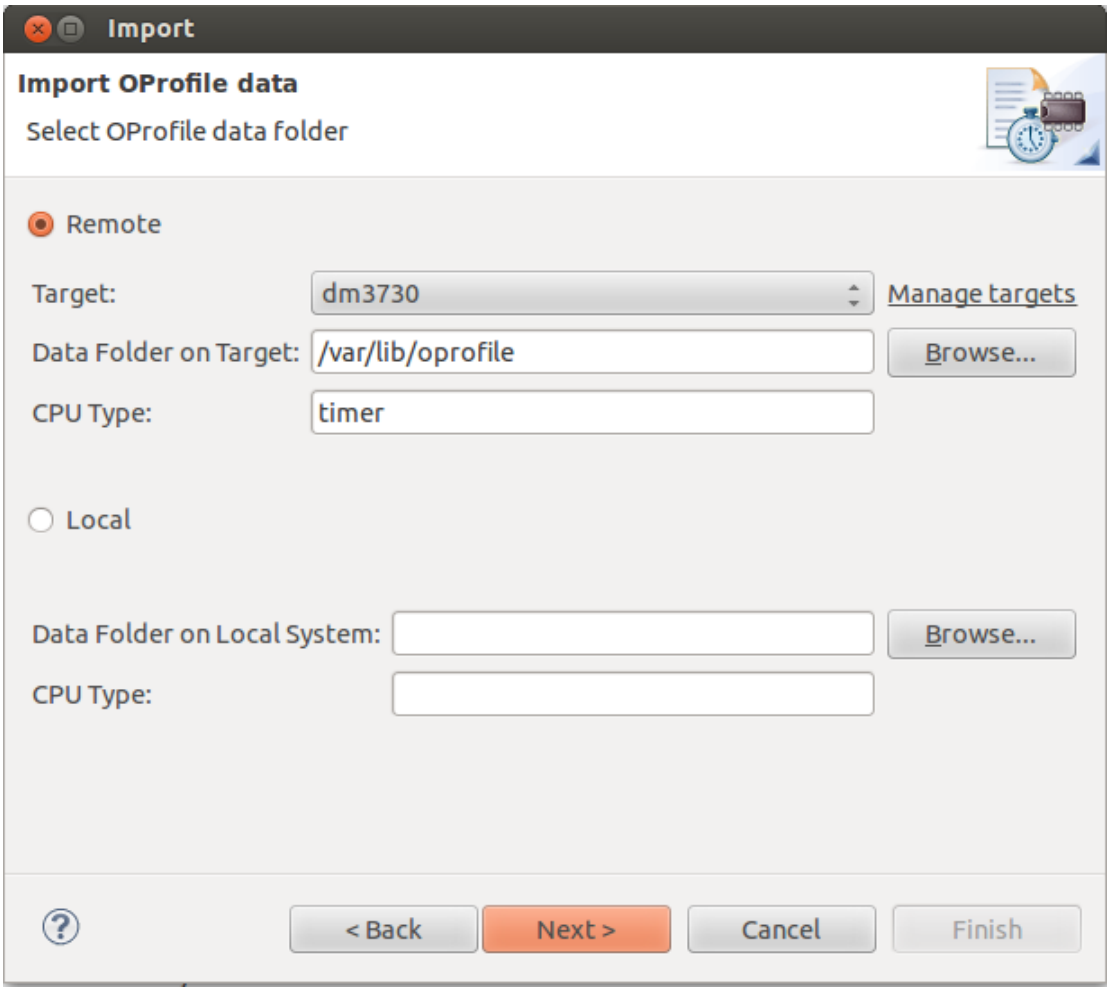


Figure 69: Import Wizard – Selecting OProfile data folder

Select either the Remote or Local radio button to import the OProfile data from a remote target or local file system, click the Browse button to select the data folder. The CPU type is detected and filled in. Click next to continue to the next page as shown in *figure 70.*

You can import the profile data into an existing project or create a new project. The target field is filled in with the remote target / local file system IP address. You may name or leave the session name blank. The timestamp is used if the session name is left blank.

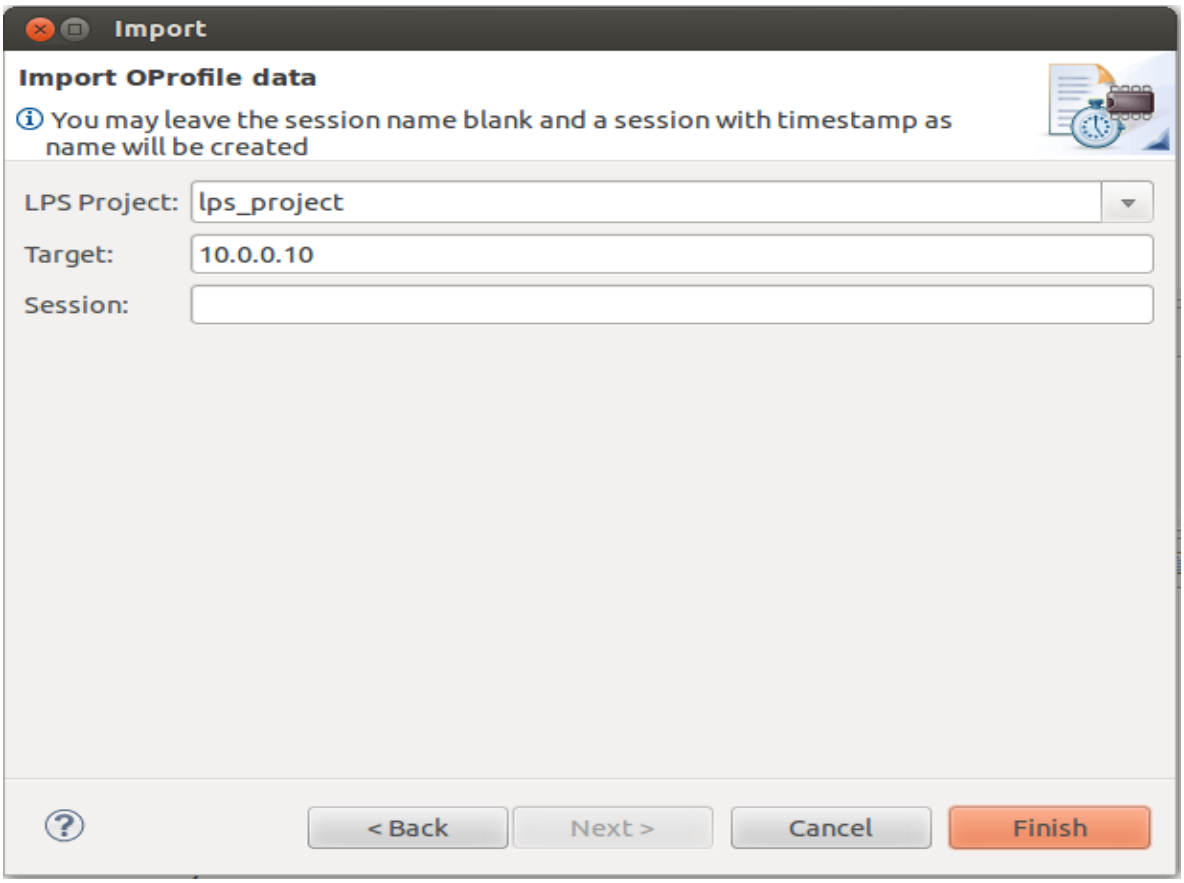


Figure 70: Import Wizard – Selecting LPS project and session

3. Click Finish to import OProfile data. The new session creation and import progress appears in the 'Profiling Monitor' view. Select this session and right-click to analyze or to view as described in previous sections (Analyzing the Profiled Data).

Comparing Profiling Sessions

During development you will profile your application and analyze the data and fix the performance bottlenecks. You will profile the application again. You can compare profile sessions in the same target by using the profile session comparator included in TimeStorm. Select the more recent session with modified code in the 'Profiling Monitor' view, right-click in the view, and select 'Compare'. The 'Compare Profile Sessions' window opens with other sessions. Select and double-click the other session

you want to compare to. The 'Compare Profile Sessions' window compares and displays the profiling results side by side as *shown in Figure 71*.

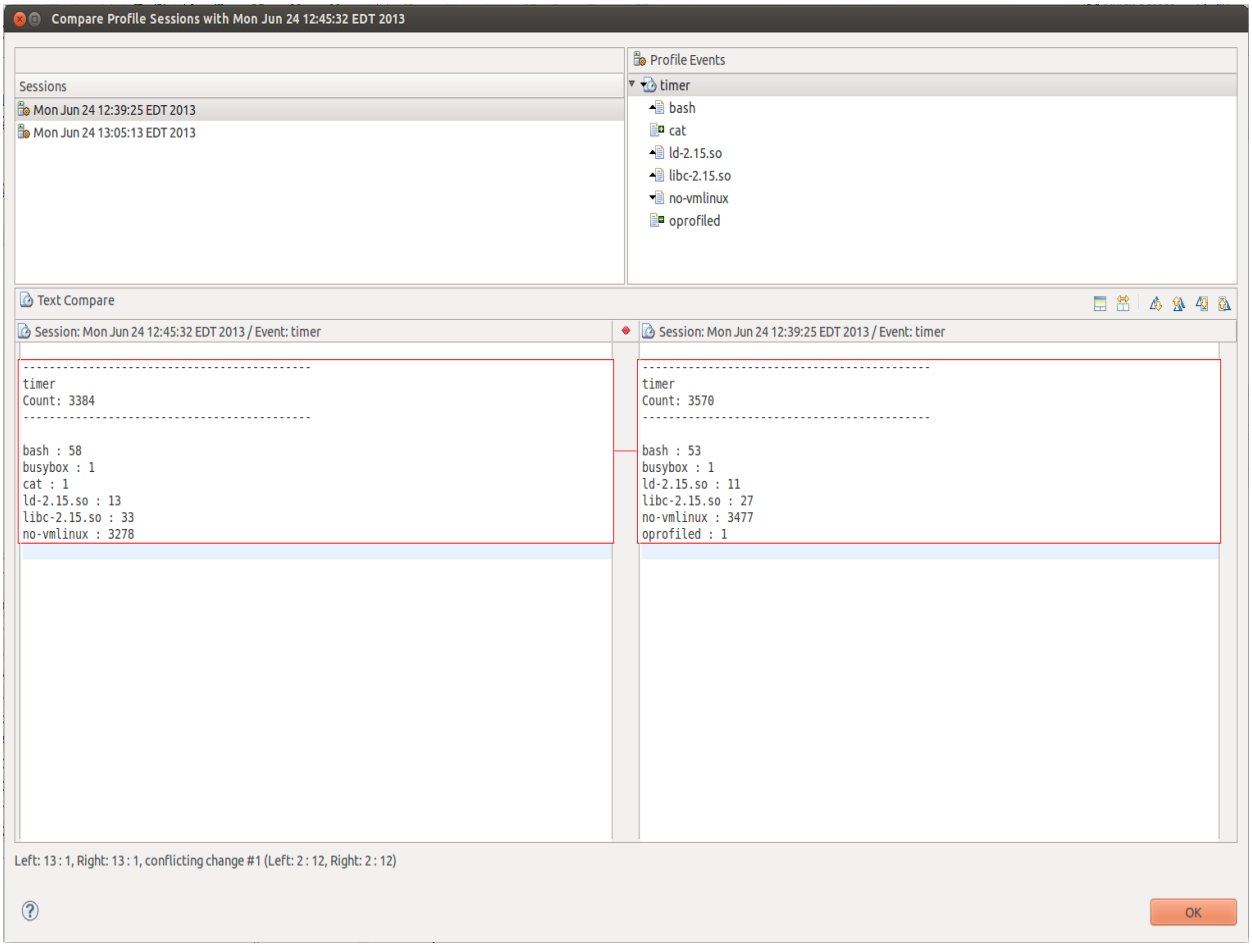


Figure 71: LPS – Comparing two sessions side-by-side

The profile events and the application / library names are displayed in the 'Profile Events' pane. You can double click on the name to just display the profile count for that application.

Valgrind Remote Support

TimeStorm adds Remote target support to the Valgrind tools from Eclipse.

Valgrind is an Instrumentation framework for building dynamic analysis tools that can be used to profile applications. Valgrind tools are used to detect memory management and threading problems.

Note: To profile using Valgrind, you should have valgrind 3.3.0 (or later) installed on your target.

Profiling using Valgrind

Refer to http://wiki.eclipse.org/Linux_Tools_Project/Valgrind/User_Guide for information of using Valgrind.

To profile on a remote target,

1. right click on the project in the project explorer, select Profiling Tools > Profiling Tools Configurations as shown in *figure 72.*

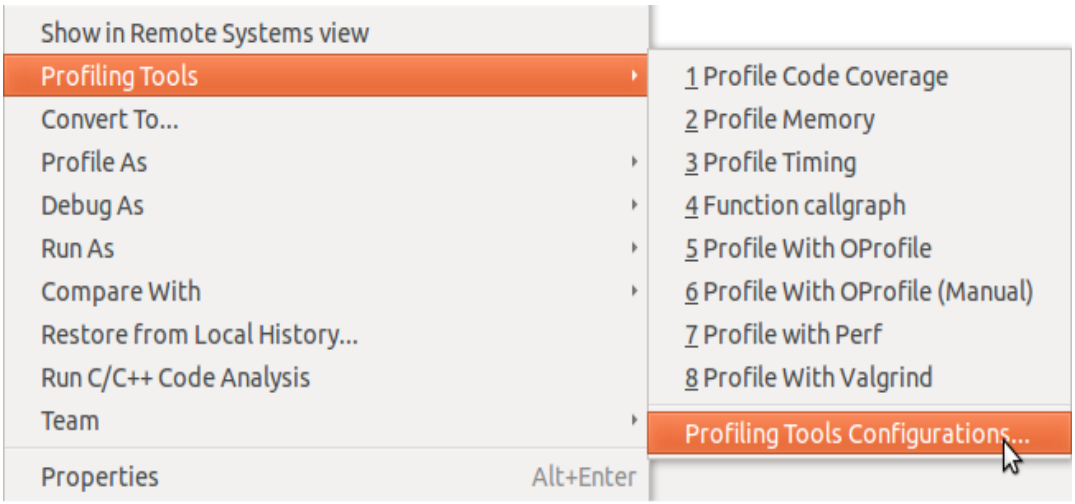


Figure 72: Launching Profile Tools Configuration

2. Select 'TimeStorm Valgrind Remote'.
3. Click New launch configuration icon in top left corner. This will create a new profile launch configuration and the launch configuration tabs are displayed in the right panel (shown in *figure 73)*
4. The launch configuration dialog has tabs as explained in http://wiki.eclipse.org/Linux_Tools_Project/Valgrind/User_Guide and an additional target tab as shown in *figure 73* to select the remote target.

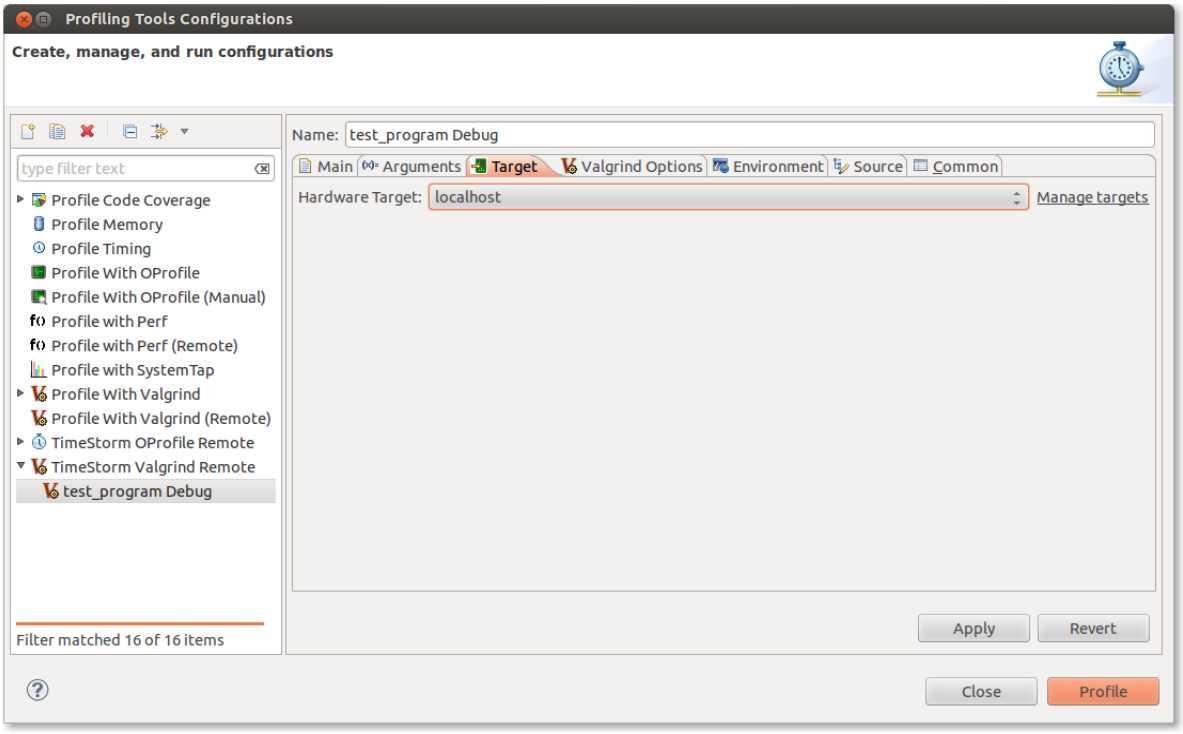


Figure 73: Creating Valgrind profile launch configuration

After you have finished configuring the settings, click the Profile button to start profiling.

LTTng

Eclipse LTTng tools bundled with TimeStorm use ssh connection to communicate to targets and hence they work with remote targets too.

Please note that LTTng tools are not integrated with the hardware targets defined in TimeStorm. You have to define the target IP address, username and password for LTTng to communicate to the target.

For help on how to use LTTng tools, please refer to
http://wiki.eclipse.org/Linux_Tools_Project/LTTng2/User_Guide

Advanced Topics

Building from the Command Line

TimeStorm has been designed so that projects can be built from the command-line as well as the IDE. This feature exists so that customers with build systems can easily integrate projects created with TimeStorm into their automated build system.

As part of the build process, TimeStorm scans the current project and builds a list of files in the project as well as the dependencies on other files in the project. TimeStorm then uses that dependency list, along with the information in the build configuration, to create several GNU make files. TimeStorm then builds the project by executing make with the generated files to perform the build steps. Each build configuration creates a make file in a directory named after the build configuration under the project directory.

For example:

```
<workspace>/my-project/Debug/
                            makefile
<workspace>/my-project/Profile/
                            makefile
```

The make file overrides the following standard variables to control what tools are used in the standard make build rules.

```
RANLIB
CPP
AS
AR
OBJCOPY
DEBUGGER
STRIP
OBJDUMP
CC
NM
CXX
LD
```

Each of these tools is changed to use the location the tool chain of the development host. If the location is different on the build machine, one of the following strategies followed:

Create symlinks — This is the easiest path. On the build machine, create symbolic links to the tools in the location where the make file expects them to be. Going this route introduces a configuration dependency on the build machine, so this should documented by the build team, and, in the best of situations, a script should be created to automate the creation of the links.

Set environment variables — With make, if an environment variable is set, it will not be overridden by a make variable. The script that runs the make file will need to set all of these variables to the right location before invoking make.

Auxiliary Files

TimeStorm will over-write the make file each time is performs a build. In order to allow for maximum customization, TimeStorm builds the make file with several optionally included files. Optionally included files are incorporated into the make file if they exist; otherwise, if the file does not exist, the reference to the file is ignored.

TimeStorm optionally includes these files:

…/makefile.init — This file is called at the beginning of the makefile. It can be used to customize initialization.

../makefile.defs — This file is called after initialization, but before objects are compiled. It can be used to supply custom macro definitions.

../makefile.targets — This file is called at the end of the makefile. It can be used to supply customized target information.

Notice TimeStorm will look for the files in the parent of the build configuration directory. This makes it easy to share make file customizations across build configurations.

Using Source Code Control from the Command-line

Some organizations use a source code control (SCC) system that can only be used from the command-line or some other external tool. TimeStorm can be used with these tools as long as care is taken in what files are placed under source control and other configuration measures are taken. This section lists out the files and directories that should not be placed under SCC and other per-file configuration information.

File or Directory/	Notes
Workspace	
<workspace>/.metadata	This directory contains state information, such as window positioning, for the current workspace. Exclude this entire directory from SCC management.
C / C++ Projects	
<workspace>/<c project>/.cproject <workspace>/<c project>/.products <workspace>/<c project>/.project	These files contain project-oriented information, such as what appears in the properties dialog for a project. Add all of these files into source control.

Embedded Linux from a Trusted Source
timesys®

About Timesys

Timesys is the provider of LinuxLink, a high-productivity software development framework that dramatically simplifies and speeds up embedded Linux application development. The LinuxLink framework includes the Linux kernel, cross-toolchain, application development IDE, an award winning build system called Factory, a vast library of middleware packages, software stacks and libraries, documentation and expert technical support. LinuxLink enables development teams to consistently build and maintain a custom, open source embedded Linux platform through regularly updated Linux sources, proven middleware packages, and a scriptable GNU-based build environment. LinuxLink reduces the time, resources, risk and cost associated with building a product based on open source Linux. For more information, visit: www.timesys.com.

Timesys Corporation
428 Forbes Avenue
Pittsburgh, PA 15219
+1 412 232 3250
+1 866 392 4897
Fax: +1 412 232 9818

TimeStorm
by timesys®